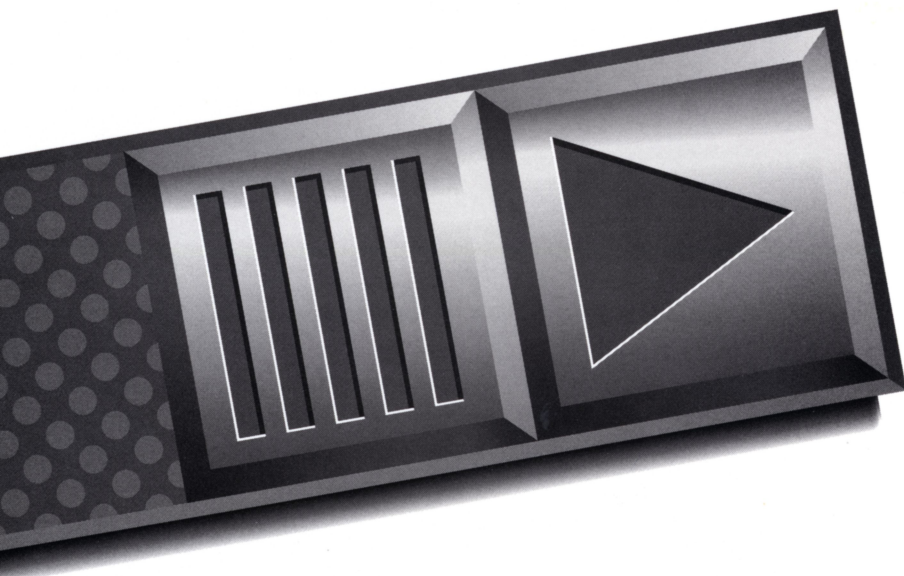
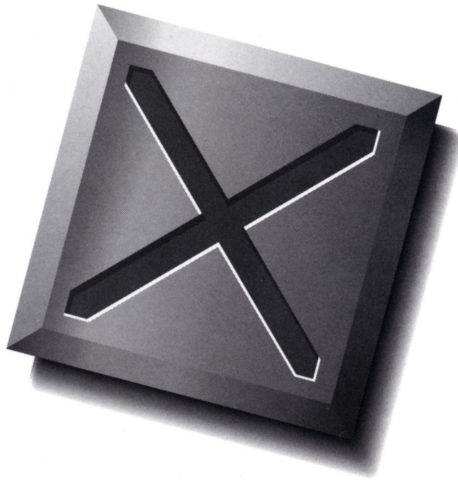


XVT



Advanced DSC++ Programming



Presented by
Mike Ernst
XVT Consulting and Training

177

1800-1801
1802-1803



Agenda

❖ General Considerations for Custom View Classes

- ♦ Consumers and adaptors
- ♦ Constraints imposed by XVT-Architect
- ♦ Composition vs. subclassing
- ♦ Opaque subviews

❖ Using CDrawingContext

- ♦ The drawing API
- ♦ Coordinate systems
- ♦ Clipping
- ♦ Logical damage - forcing a redraw



Advanced XVT-Power++

Slide 1



Agenda (cont.)

❖ Implementing a Simple Custom View

- ♦ Writing a Draw() function
- ♦ Responding to size changes

❖ Designing a Custom Subview

- ♦ Analyzing roles and responsibilities
- ♦ Designing for both consumers and adaptors
- ♦ Defining notification commands
- ♦ Managing contents

❖ Implementing a Custom Subview

- ♦ Subclassing options
- ♦ Public interface vs actual implementation
- ♦ Optimizing a Draw() function

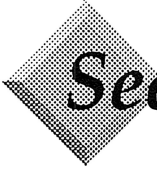


Advanced XVT-Power++

Slide 2

Section 1

Writing and Using Custom View Classes



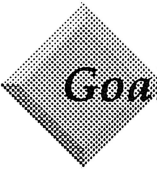
Section 1

Writing and Using Custom View Classes



Advanced XVT-Power++

Slide 1



Goals of Section 1

- ❖ Distinguish “consumers” and “adaptors”
- ❖ Understand how to use the user-view and user-subview features of XVT-Architect
- ❖ Identify design considerations when writing custom subview and view classes
- ❖ Review frame concepts and coordinate transformations
- ❖ Gain experience using CDrawingContext



Advanced XVT-Power++

Slide 2

Consumers and Adaptors

- ❖ A novice to intermediate user of a framework is sometimes referred to as a “consumer”
 - ♦ The screens they implement use view classes directly from the framework, with methods overridden typically to add behavior when the user interacts with those views
- ❖ A more experienced user of a framework often takes an additional role of “adaptor”
 - ♦ An adaptor extends the functionality provided by the framework, by writing new view classes, for example, or by changing the way in which input events are handled
 - ♦ The key to success here is to extend the functionality without modifying the base classes
- ❖ This course will help you be a proficient adaptor

Advanced XVT-Power++

Slide 3

CUIView and CUserSubview

- ❖ XVT-Architect’s Drafting Board palettes support user-defined view or subview classes
 - ♦ The factory code will instantiate and initialize your own custom views and subviews
- ❖ You must provide a constructor for your class that has only two required arguments - theEnclosure and theRegion

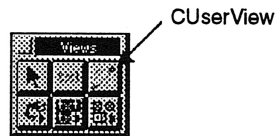
```
XMyView1:: XMyView1(CSubview* theEnclosure,  
                  const CRect& theRegion);  
XMyView2:: XMyView2(CSubview* theEnclosure,  
                  const CRect& theRegion,  
                  long theSpecialCommand = NULLcmd, ... );
```

Advanced XVT-Power++

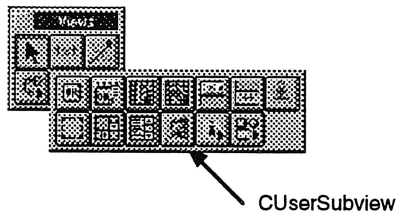
Slide 4

Placing Custom Views

- ❖ The User-View tool is on the Views palette



- ❖ The User-Subview tool is on the Subviews palette

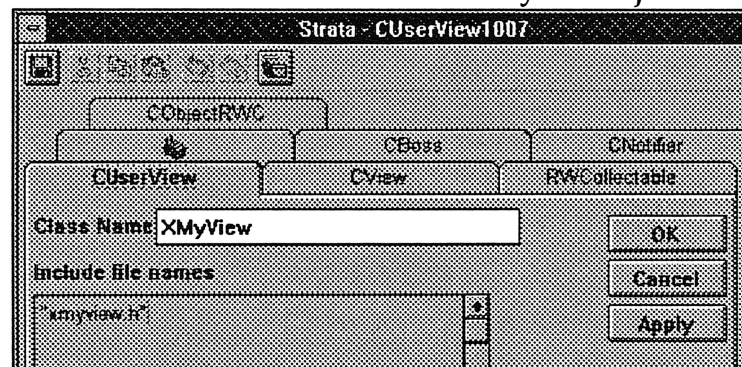


Advanced XVT-Power++

Slide 5

Identifying Your Custom Class

- ❖ The factory needs to know the class name and the include files needed to instantiate your object



Advanced XVT-Power++

Slide 6

Initializing Custom Views

❖ Define many properties with XVT-Architect

- ♦ Size and location
- ♦ Single and double commands
- ♦ Glue
- ♦ Draggable and/or sizeable
- ♦ Environment settings
- ♦ Factory name

❖ Set derived-class-specific attributes in code

```
itsData.itsMySpecialView->ISpecialView( DRAGGINGcmd, DROPcmd );
```

Advanced XVT-Power++

Slide 7

Writing a Custom View Class

❖ You must implement a Draw() function

- ♦ Units - best to use default (pixel) units

```
virtual void Draw(const CRect& theClippingRegion);
```

❖ Remember to override the Size() method so Glue behavior will work properly

- ♦ Recalculate size and position of each drawing component
- OR -
- ♦ Recalculate the mapping from logical to physical units

```
void XGraph::Size( const CRect& theNewSize )  
{  
    // Call base class method to change the size of the frame,  
    // then recalculate the mapping of each drawn point  
    CView::Size( theNewSize );  
    MapPoints( );  
}
```

Advanced XVT-Power++

Slide 8

Writing a Custom Subview Class

- ❖ **When designing a custom subview, consider...**
 - ♦ Will the user be able to interact with the nested views (dragging the lines of a graph, for instance)
 - ♦ Will the user interact with the subview as a unit - if so, you need to override `FindEventTarget()` and `FindDeepSubview()`
 - ♦ Do you always want downward chaining to work? You may want to override `DoSetGlue()`, `DoSetEnvironment()`, etc. to break the downward chaining

Advanced XVT-Power++

Slide 9

Opaque Subviews

- ❖ **In some cases, you need a view that owns other views, but you want the input handling to be determined by the outer view**
 - ♦ The name given to this subview is an "opaque subview"
- ❖ **To implement opacity, override `FindEventTarget`, and `FindDeepSubview` to not search the object hierarchy**

```
inline CView* XMyOpaqueSubview::FindEventTarget(  
                                const CPoint& theLocation ) const  
{ return (CView*) this; }  
  
inline CView* XMyOpaqueSubview::FindDeepSubview(  
                                const CPoint& theLocation ) const  
{ return (CView*) this; }
```

Advanced XVT-Power++

Slide 10

Drawing Custom Views - Options

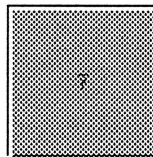
- ❖ **Create CView objects for each component of the drawing**
 - ♦ The components will draw themselves, can have independent environments, generate commands and be dragged or sized
 - ♦ Results in a heavy-weight subview object - only do this if you need to interact with the components of the drawing (dragging, sizing or command generation)
 - ♦ A good technique for prototyping
- ❖ **Write a Draw() function using CDrawingContext**
 - ♦ Slightly more programming work
 - ♦ Results in a light-weight, efficient view
- ❖ **Remember, you also have access to the PTK or native SDK layers, if desired, in Draw()**

Advanced XVT-Power++

Slide 11

An Example

- ❖ **Consider a draggable tile for a shuffle board game**



- ❖ **It could be implemented as an opaque subview**
 - ♦ Good prototyping technique
- ❖ **It could also be implemented as a CView class**
 - ♦ There is no real need to interact with the components of the tile object

Advanced XVT-Power++

Slide 12

An Opaque Subview

```
class XTile : public CRectangle {
public:
    XTile(CSubview *theEnclosure,
          const CRect& theRegion,
          const CStringRW& theTitle,
          COLOR theTileColor = COLOR_CYAN,
          COLOR theLabelColor = COLOR_DKGRAY,
          COLOR theBorderColor = COLOR_LTGRAY,
          COLOR theHiLiteColor = COLOR_WHITE,
          COLOR theShadowColor = COLOR_GRAY,
          int the3DThickness = 2);
    virtual ~XTile( void ) {}
    // Override these so the tile gets dragged as a unit
    virtual CView* FindEventTarget(const CPoint& theLocation) const
        { return (CView*)this; }
    virtual CView* FindDeepSubview(const CPoint& theLocation) const
        { return (CView*)this; }
    // Intentionally break downward chaining
    virtual void DoSetGlue(GLUETYPE theNewGlue)
        { SetGlue(theNewGlue); }
};
```

Advanced XVT-Power++

Slide 13

An Opaque Subview

❖ Create the components of the tile using CViews

```
XTile::XTile(... args ...)
: CRectangle(theEnclosure, theRegion)
{
    ... Setup the environment and calculate geometry objects ...
    // create a label for the tile
    CText* aText = new CText( this, anInsetFrame, theTitle );
    aText->SetPlacement( CText::CENTER );
    aText->SetGlue( ALLSTICKY );

    // create shadows on the bottom and right for a 3D effect
    anEnv.SetPen( itsShadowColor, its3DThickness, PAT_SOLID );
    CLine* aLine = new CLine( this, rightTop, rightBottom );
    aLine->SetGlue( TOPRIGHTSTICKY|BOTTOMSTICKY );
    aLine->SetEnvironment( anEnv );

    aLine = new CLine( this, leftBottom, rightBottom );
    aLine->SetGlue( BOTTOMLEFTSTICKY | RIGHTSTICKY );
    aLine->SetEnvironment( anEnv );
    ...
}
```

Advanced XVT-Power++

Slide 14

An Opaque Subview (cont.)

```
XTile::XTile(... args ...)
: CRectangle(theEnclosure, theRegion)
{
    ....
    // finish the 3D effect with hilights on the left and top
    anEnv.SetPenColor( itsHiLiteColor );
    aLine = new CLine( this, leftTop, leftBottom );
    aLine->SetGlue( TOPLEFTSTICKY|BOTTOMSTICKY );
    aLine->SetEnvironment( anEnv );

    aLine = new CLine( this, leftTop, rightTop );
    aLine->SetGlue( TOPLEFTSTICKY | RIGHTSTICKY );
    aLine->SetEnvironment( anEnv );
}
```

Advanced XVT-Power++

Slide 15

The Drawing Context Class

❖ CDrawingContext encapsulates a drawing API

- ♦ Use it when you want to write custom view classes with specialized visual appearance, but the components of the view don't themselves need CView capabilities

❖ Examples

- ♦ Graph views that are display-only
- ♦ Drawing beveled edges for a 3-D appearance
- ♦ Displaying bitmaps aligned with text in a list control
- ♦ Lightweight text drawing for labels on a drawing or large amounts of text in a grid
- ♦ Many others ...

Advanced XVT-Power++

Slide 16

CDrawingContext Features

- ❖ Provides an API that uses CPoint, CRect and CEnvironment data types
 - ♦ All coordinates must be global (window-relative)
- ❖ Manages drawing tools (environment)
 - ♦ Makes sure that the environment information is set properly each time the drawing context comes in to scope
- ❖ Units management
 - ♦ To communicate with the PTK a conversion is needed from CRect and CPoint to RCT and PNT. The Power++ conversion operators use global units. The drawing context makes sure SetGlobalUnits has been called properly.

 Advanced XVT-Power++

Slide 17

CDrawingContext API

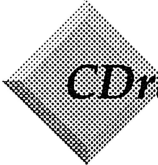
- ❖ Constructors
 - ♦ Create a drawing context to draw in a view's CWindow using the units and environment of a view

```
CDrawingContext(CView *theView);
```
 - ♦ Create a drawing context to draw in a specified CWindow using the window's units and the specified environment

```
CDrawingContext(CWindow *theWindow,  
                const CEnvironment &theEnvironment);
```

 Advanced XVT-Power++

Slide 18



CDrawingContext API (cont.)

❖ Methods for drawing lines and open shapes

```
void DrawLine(const CPoint &theStartPoint,
              const CPoint &theEndPoint,
              BOOLEAN theBegArrow = FALSE,
              BOOLEAN theEndArrow = FALSE);

void SetLinePos(const CPoint &theLoc);
void DrawLine(const CPoint &theEndPoint,
              BOOLEAN theBegArrow = FALSE,
              BOOLEAN theEndArrow = FALSE);

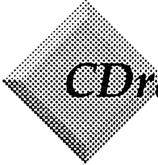
void DrawArc(const CRect &theFrame,
             const CPoint &theStartPnt,
             const CPoint &theEndPnt);

void DrawPolygon(const RWOordered &thePoints,
                 BOOLEAN isFilled = TRUE); // use FALSE for open shape
```



Advanced XVT-Power++

Slide 19



CDrawingContext API (cont.)

❖ Methods for drawing basic closed shapes

```
void DrawPie(const CRect &theFrame,
             const CPoint &theStartPnt,
             const CPoint &theEndPnt);

void DrawPolygon(const RWOordered &thePoints,
                 BOOLEAN thePolyIsFilled = TRUE);

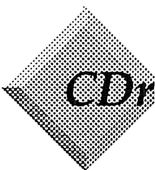
void DrawRect(const CRect &theFrame,
              short theWidthRounded = 0, // w & h in pixels
              short theHeightRounded = 0);

void DrawOval(const CRect &theFrame);
```



Advanced XVT-Power++

Slide 20



CDrawingContext API (cont.)

❖ **Methods for drawing 3-D shapes**

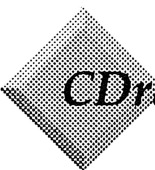
```
void DrawOutsetFrame(const CRect &theOuterFrame,
    int             theBorderWidth, /* in pixels */
    COLOR           theUpShadow,
    COLOR           theDownShadow);

void DrawInsetFrame(const CRect &theOuterFrame,
    int             theBorderWidth, /* in pixels */
    COLOR           theUpShadow,
    COLOR           theDownShadow);
```



Advanced XVT-Power++

Slide 21



CDrawingContext API (cont.)

❖ **Methods for drawing text and managing the caret**

```
void DrawText(const CStringRW &theText,
    const CPoint &theBaseLine);

void SetCaretPos(const CPoint &theLoc);
void SetCaretVisible(BOOLEAN isVisible = TRUE);
void SetCaretSize(int theWidth,
    int theHeight); // w & h in pixels
```



Advanced XVT-Power++

Slide 22



CDrawingContext API (cont.)

❖ **Methods for displaying bitmaps, images and icons**

```
void DrawIcon(int theIconId,
              const CPoint &theTopLeft);

void DrawPicture(PICTURE thePict,
                 const CRect &theViewRegion);

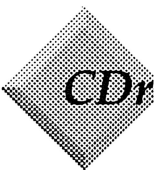
void DrawPixmap(XVT_PIXMAP thePixmap,
                const CRect &theSourceRegion,
                const CRect &theDestRegion);

void DrawImage(XVT_IMAGE theImage,
               const CRect &theSourceRegion,
               const CRect &theDestRegion);
```



Advanced XVT-Power++

Slide 23



CDrawingContext API (cont.)

❖ **Methods for clipping region and environment management**

```
void SetEnv(const CEnvironment &theEnvironment);
CEnvironment & GetEnv(void);
CEnvironment & operator () (void) { return GetEnv(); }

void SetClip(const CRect &theClipFrame = MAXRect);
CRect GetClip(void);
```



Advanced XVT-Power++

Slide 24

CDrawingContext API (cont.)

- ❖ **Determine if a region needs to be re-drawn**

```
BOOLEAN IsUpdateNeeded(const CRect &theRegion);
```

- ❖ **Mark a region as needing to be re-drawn - i.e. cause logical damage to a part of the view**

```
void InvalidateRect(const CRect &theRegion = MAXRect);
```

- ❖ **Move the pixels of a region by a horizontal and vertical amount**

```
void ScrollRect(const CRect &theRegion,  
               int theHDist,  
               int theVDist);
```

Advanced XVT-Power++

Slide 25

Optimizing Drawing

- ❖ **To ensure you only draw the regions of views that need to be redrawn, call CView::Prepare()**

- ♦ Sets up the clipping region
- ♦ Returns a BOOLEAN value indicating whether the region needs to be drawn in

```
void XMyCustomView::Draw(const CRect &theDamageRegion)  
{  
    // draw the view only if it intersects the  
    // region needing to be redrawn  
    if (Prepare(theDamageRegion - GetGlobalFrame())) {  
        ... drawing commands for this view ...  
    }  
}
```

Advanced XVT-Power++

Slide 26

Using CDrawingContext

- ❖ The Draw() function in your custom view class might look something like this...

```
void XGraph::Draw(const CRect &theDamageRegion)
{
    // Use standard CView method to setup clipping and make sure
    // this view lies within the area needing to be redrawn
    if (Prepare(theDamageRegion - GetGlobalFrame())) {
        CDrawingContext aDC(this); // setup environment and units

        // Draw a polygon connecting the points of the graph. The
        // interior of the polygon will be filled using the brush
        aDC.DrawPolygon(itsDrawnPoints);

        // Draw the title at the top-left of the view
        aDC.DrawText(itsTitle, CPoint(10,20));
    }
}
```

Globalize(this)

Advanced XVT-Power++

Slide 27

The Tile Example Revisited

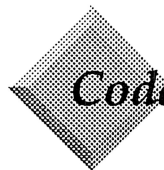
```
void XTile::Draw(const CRect &theDamageRegion)
{
    if (Prepare(theDamageRegion - GetGlobalFrame())) {
        CDrawingContext aDC(this); // setup environment and units

        // Draw a 3D Tile
        aDC.DrawOutsetFrame(GetGlobalFrame(),
                           the3DThickness,
                           theHiliteColor,
                           theShadowColor );

        // Draw the title centered in the frame
        CPoint aLocation( ...);
        aDC.DrawText(itsTitle, aLocation);
    }
}
```

Advanced XVT-Power++

Slide 28



Code Walkthrough

- ❖ Implement a custom view class that takes a set of points and graphs them, either as a basic line graph or bar chart

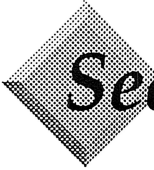


Advanced XVT-Power++

Slide 29

Section 2

Building a Custom Subview



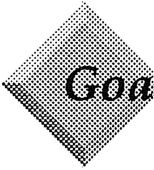
Section 2

Building a Custom Subview



Advanced XVT-Power++

Slide 1



Goals of Section 2

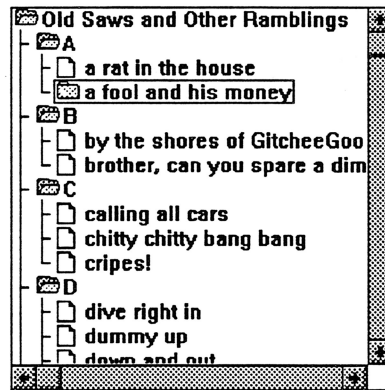
- ❖ **Develop a thought process for designing a custom subview class**
 - ♦ **Basic OO Analysis and Design**
 - Define requirements
 - Translate requirements into an object model
 - Determine object roles and responsibilities
 - Defining the class interface
 - ♦ **Power++-specific considerations**
 - Frame concepts
 - Subclassing options



Advanced XVT-Power++

Slide 2

A Hierarchy Browser Subview



Advanced XVT-Power++

Slide 3

Browser Requirements

- ❖ Display a vertical hierarchy of information, descended from a root node
- ❖ Hierarchy can scroll horizontally and vertically
- ❖ The hierarchy can be single-select or multi-select
- ❖ The hierarchy may display lines connecting a parent node to its children
- ❖ Nodes are labelled with text, and possibly with an image
- ❖ Nodes can be dragged and dropped in new locations in the hierarchy

Advanced XVT-Power++

Slide 4

Designing for Reuse

- ❖ **Wear two different “hats” during the design process**
 - ♦ Consumer - think from the viewpoint of the programmer who will use the subview class
 - ♦ Adaptor - also consider how another programmer may want to extend or enhance your functionality
- ❖ **For the hierarchy browser, the design should allow for enhancements**
 - ♦ horizontal instead of vertical orientation
 - ♦ organization chart format
 - ♦ PERT charts
 - ♦ outlining

Advanced XVT-Power++

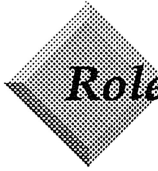
Slide 5

Object Analysis

- ❖ **What are the objects that arise from analyzing the requirements?**
 - ♦ Nodes display the label and possibly image information
 - ♦ A subtree is a node and all its child nodes
 - ♦ A hierarchy owns the root node and allows all the nodes to be displayed, providing scrolling if necessary
- ❖ **How do we translate this into Power++ classes?**
 - ♦ XHierarchyBrowser - a subview
 - ♦ XSubtree - a subview to represent a single node or an entire branch of the hierarchy

Advanced XVT-Power++

Slide 6



Roles and Responsibilities

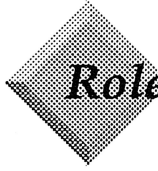
❖ XHierarchyBrowser

- ♦ Owns the root subtree
- ♦ Keeps track of the size of the root subtree
- ♦ Implements the scrolling behavior
- ♦ Manages scrolling ranges
- ♦ Draws connecting lines (or not) between the nodes
- ♦ Implements the single or multi-select behavior
- ♦ Determines if and when a subtree is being dragged
- ♦ Others...?



Advanced XVT-Power++

Slide 7



Roles and Responsibilities (cont.)

❖ XSubtree

- ♦ Draws the node information (text, images, etc)
- ♦ Manages frame concepts for the subtree
- ♦ Implements the subtree drop behavior
- ♦ Generates a notification when a node is selected, deselected or dropped
- ♦ Computes and maintains layout information based on the assigned font
- ♦ Others...?

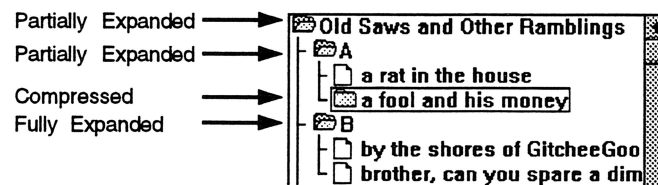


Advanced XVT-Power++

Slide 8

Frame Concepts

- ❖ A subtree can be in one of 3 states
 - ♦ Compressed
 - ♦ Partially expanded
 - ♦ Fully expanded
- ❖ Each of these states will be represented by a different frame concept



Advanced XVT-Power++

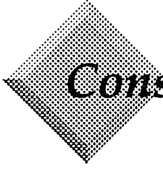
Slide 9

Frame Concepts (cont.)

- ❖ For ease of implementation, the frame should be the dimensions of the subtree at its full expansion
- ❖ Add two new frame interfaces
 - ♦ Minimum Frame - the dimensions of the compressed subtree
 - ♦ Current Frame - the dimensions of the subtree at its current level of expansion
- ❖ Other standard CView frame concepts
 - ♦ Clipped Frame - contains the region of the view clipped to all its enclosures - not good to use in a virtual frame
 - ♦ Visible Frame - the region of the view at its minimum state - has meaning for NListEdit and NListButton. A possible alternative to the MinimumFrame concept

Advanced XVT-Power++

Slide 10



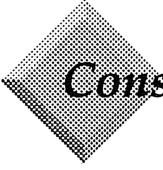
Consider the Consumer

- ❖ Does the consumer need to know about both classes, the subtree node and the hierarchy?
 - ♦ The consumer should have enough of an API to be able to get information about a node, but can't create nodes directly
- ❖ Who should manage the data?
 - ♦ Ease of use
 - the hierarchy object is initialized with all the data
 - capacity limited to available memory
 - ♦ Unlimited capacity
 - the application programmer manages the information
 - more programming required to use the object
 - more programming required to implement the class



Advanced XVT-Power++

Slide 11



Consider the Adaptors

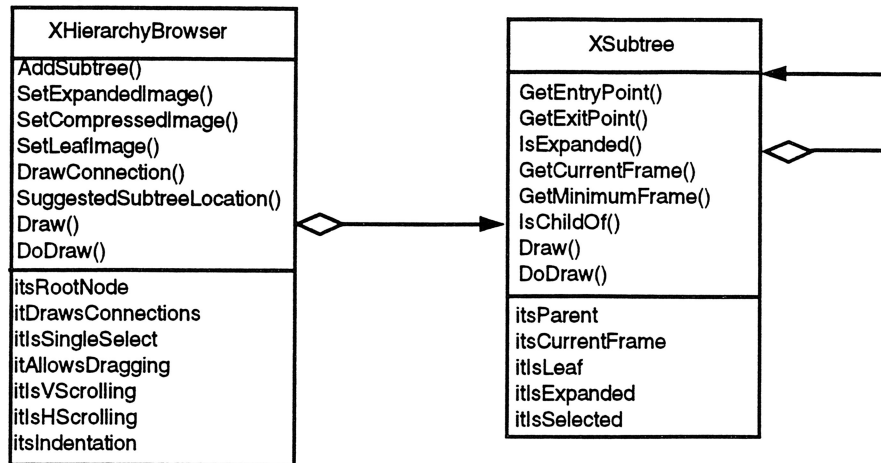
- ❖ How should the design reflect extensibility requirements?
 - ♦ PERT charts, organization charts, vertical or horizontal hierarchy, outlines?
- ❖ Three additional features seem key
 - ♦ If the hierarchy object computes the initial location of a subtree node, a derived hierarchy class can implement a different layout policy
 - ♦ If the subtree node provides an interface that identifies where the connecting lines attach to the node, the hierarchy object can adjust to different node types and still draw the connections
 - ♦ To accomodate different connection line styles, the hierarchy should encapsulate the line drawing into a virtual function



Advanced XVT-Power++

Slide 12

Class Diagram



Advanced XVT-Power++

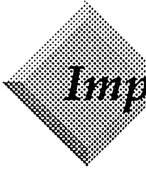
Slide 13

API vs Implementation

- ❖ The public interface to the hierarchy browser
 - ◆ Provides the ability to modify display properties of the browser
 - ◆ Provides the interface to add nodes and set node properties
 - ◆ Defines the notification commands that the consumer will receive
- ❖ The public interface to the subtree nodes
 - ◆ Allows the consumer to inquire a node's properties, but not to modify them

Advanced XVT-Power++

Slide 14



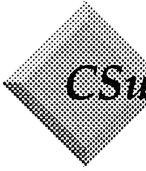
Implementing the Hierarchy Class

- ❖ How do we translate the object design for XHierarchyBrowser and XSubtree into CView or CSubview classes?
- ❖ Subclassing options for XHierarchyBrowser
 - ♦ CSubview
 - ♦ CGrid
 - ♦ CCTable
 - ♦ CScroller



Advanced XVT-Power++

Slide 15



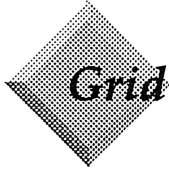
CSubview Properties

- ❖ Useful Properties for XHierarchyBrowser
 - ♦ Provides the ability to own and manage subtree nodes
 - ♦ Virtual Draw and DoDraw functions can be overridden to draw nodes and hierarchy lines
- ❖ Drawbacks
 - ♦ Too much work to write geometry calculations and scrolling; higher level classes can be used



Advanced XVT-Power++

Slide 16



Grid Properties

❖ Useful properties

- ♦ Implemented as opaque subview
- ♦ All layout calculations done for us
- ♦ Using a variable grid allows easy implementation of resize and redraw when a subtree changes from compressed to expanded or vice versa
 - The node size changes, then call AdjustRow or AdjustCol with a MAXIMIZE adjustment policy

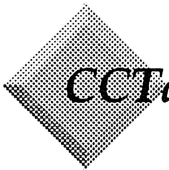
❖ Drawback

- ♦ Adaptors may not want to arrange the nodes according to a rigid matrix



Advanced XVT-Power++

Slide 17



CCTable Properties

❖ Useful properties

- ♦ Aligns information in vertical columns
- ♦ Allows drawing as well as text
- ♦ Implements multiple buffering styles - solves both the ease of use and the unlimited capacity tradeoff

❖ Drawback - the rows of a table are not views

- ♦ Difficult to add in drag and drop behavior
- ♦ Difficult for adaptors to extend to either horizontal or vertical format



Advanced XVT-Power++

Slide 18

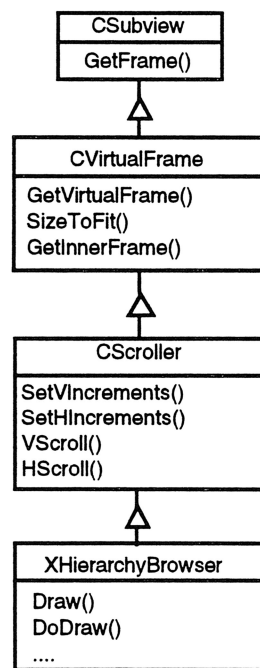
CScroller Properties

- ❖ CScroller implements two essential features of the hierarchy browser
 - ♦ Virtual frame behavior - able to manage and display a very large number of subtree nodes
 - ♦ Scrolling of nested views
- ❖ This is the best class to derive the behavior from

Advanced XVT-Power++

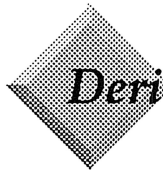
Slide 19

Inheritance



Advanced XVT-Power++

Slide 20



Deriving from CScroller

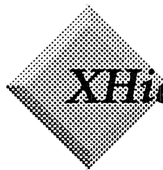
❖ What functionality do we need to implement?

- ♦ Constructors and destructor
- ♦ Initializer function
- ♦ Function to suggest a location for a new node
- ♦ Consumer API functions to add nodes to the tree and to control the display properties
- ♦ Override Draw() to display lines connecting the nodes
- ♦ Add logic to adjust the scrolling behavior when subtrees are expanded and contracted, when the size changes, or when the font is changed



Advanced XVT-Power++

Slide 21



XHierarchyBrowser Class

XHierarchyBrowser
AddSubtree() SetExpandedImage() SetCompressedImage() SetLeafImage() DrawConnection() SuggestedSubtreeLocation() Draw() DoDraw()
itsRootNode itDrawsConnections itIsSingleSelect itAllowsDragging itIsVScrolling itIsHScrolling itsIndentation



Advanced XVT-Power++

Slide 22

Constructors

- ❖ Remember that Architect requires a constructor with only two required arguments

```
XHierarchyBrowser::XHierarchyBrowser( CSubview* theEnclosure,
                                     const CRect& theRegion,
                                     UNITS theIndentPerLevel,
                                     BOOLEAN isDrawingConnectingLines,
                                     BOOLEAN isSingleSelect,
                                     BOOLEAN isDragAndDropEnabled,
                                     BOOLEAN isHScrolling,
                                     BOOLEAN isVScrolling )
: CScroller( theEnclosure, theRegion ),
  itsIndentation( theIndentPerLevel ),
  itsRoot( 0 ),
  itDrawsConnections( isDrawingConnectingLines ),
  itIsSingleSelect( isSingleSelect ),
  itAllowsDragging( isDragAndDropEnabled ),
  itIsHScrolling( isHScrolling ),
  itIsVScrolling( isVScrolling )
{ } // Also implement copy constructor and assignment operator
```

Advanced XVT-Power++

Slide 23

An Initializer Function

- ❖ Stylistically, all view classes should have an initializer

```
BOOLEAN XHierarchyBrowser::IHierarchyBrowser(
    BOOLEAN isHScrolling,
    BOOLEAN isVScrolling,
    BOOLEAN isDrawingConnectingLines,
    BOOLEAN isSingleSelect,
    BOOLEAN isDragAndDropEnabled )
{ IScroller( isHScrolling, isVScrolling, TRUE,
             10, 50, // Values will be changed by ReSynch()
             TRUE, GetGlue() );
  itIsHScrolling = isHScrolling;
  itIsVScrolling = isVScrolling;
  itDrawsConnections = isDrawingConnectingLines;
  itIsSingleSelect = isSingleSelect;
  itAllowsDragging = isDragAndDropEnabled;
  ReSynch();
  return TRUE;
}
```

Advanced XVT-Power++

Slide 24

New Node Location

- ❖ If the hierarchy class supplies a location for each new subtree, adaptors can derive to display other hierarchy styles

```
CPoint XHierarchyBrowser::SuggestedSubtreeLocation(
                                const XSubtree* theParent )
{
    CPoint theUpperLeft(HMargin,VMargin);
    if (theParent) {
        // For a vertical hierarchy, suggest a location at the
        // bottom of the existing children of this node
        UNITS aHeight = theParent->GetMinimumFrame().Height() +
                        theParent->CurrentHeightOfChildren();
        theUpperLeft.V( HMargin + aHeight );
        theUpperLeft.H( VMargin + itsIndentation );
    }
    return theUpperLeft;
}
```

Advanced XVT-Power++

Slide 25

Consumer API

- ❖ The user of the class will add subtree nodes through the API provided by the browser class

```
XSubTree* XHierarchyBrowser::AddSubtree(XSubTree* theParent,
                                const CStringRW& theLabel,
                                BOOLEAN isExpanded )
{
    XSubTree *aNewNode;
    aNewNode = new XSubTree( theParent ?
                            (CSubview*)theParent : (CSubview*)this,
                            SuggestedSubtreeLocation(theParent),
                            theParent,
                            theLabel,
                            isExpanded );

    if (aNewNode) {
        if (theParent)
            theParent->AddedChild( aNewNode );
        else // TODO - check for replacing root node
            itsRoot = aNewNode;
    }
    return aNewNode;
}
```

Advanced XVT-Power++

Slide 26

Consumer API (cont.)

- ❖ Another important aspect of a custom view class is the type of command notifications it will generate
 - ♦ Are the standard single command and double command appropriate for this view?
 - ♦ Are additional notifications needed?
- ❖ For the hierarchy browser, the consumer may want to be notified...
 - ♦ When a subtree is selected
 - ♦ When a subtree is deselected
 - ♦ When a subtree is moved
 - ♦ When a subtree is expanded or compressed

Advanced XVT-Power++

Slide 27

Draw() Function

- ❖ Draw() extends the default CScroller behavior by possibly drawing lines connecting the nodes

```
void XHierarchyBrowser::Draw( const CRect& theClippingRegion )
{
    CScroller::Draw( theClippingRegion );
    CDrawingContext aDC( this );
    if (itDrawsConnections && itsRoot->IsExpanded())
        // Draw lines from parent to child, recursing down the tree
        ConnectToChildren( itsRoot, aDC, theClippingRegion );
}
```

Advanced XVT-Power++

Slide 28

Connecting the Nodes

- ❖ The function to connect a parent to its children calls a virtual function to draw each connection

- ♦ Notice that the node tells the browser where to draw the lines

```
void XHierarchyBrowser::ConnectToChildren(
    const XSubtree* theParent,
    CDrawingContext& theDC,
    const CRect& theRegion )
{
    XSubtree* aChild;
    CPoint anEndPoint, aStartPoint;
    aStartPoint = theParent->GetExitPoint().Globalize(theParent);

    RWOIterator nextChild( *theParent->GetSubviews() );
    while ((aChild = (XSubtree*)nextChild()) != NULL) {
        anEndPoint = aChild->GetEntryPoint().Globalize(aChild );
        DrawConnection( theDC, aStartPoint, anEndPoint );
        if (aChild->IsExpanded())
            ConnectToChildren( aChild, theDC, theRegion );
    }
}
```

Advanced XVT-Power++

Slide 29

Drawing the Connections

- ❖ The virtual function to connect the parent to the child draws a right angle connection, but can be overridden by an adaptor of the class

```
void XHierarchyBrowser::DrawConnection(
    CDrawingContext& theDC,
    const CPoint& theStartPoint,
    const CPoint& theEndPoint )
{
    CPoint aMiddlePoint( theStartPoint.H(), theEndPoint.V() );

    theDC.DrawLine( theStartPoint, aMiddlePoint );
    theDC.DrawLine( theEndPoint, aMiddlePoint );
}
```

Advanced XVT-Power++

Slide 30

Optimizing the Drawing

- ❖ After the initial implementation, add logic to stop drawing lines outside the clipping region

```
void XHierarchyBrowser::ConnectToChildren( .... )
{
    .... compute start point ....
    if (aStartPoint.H() > theRegion.Right() ||
        aStartPoint.V() > theRegion.Bottom()) return;

    RWOIterator nextChild( *theParentNode->GetSubviews() );
    while ((aChild = (XSubtree*)nextChild()) != NULL) {
        .... compute end point ....
        if (anEndPoint.V() > theRegion.Top() &&
            anEndPoint.H() > theRegion.Left())
            DrawConnection( theDC, aStartPoint, anEndPoint );
        const RWOIterator* nestedChildren = aChild->GetSubviews();
        if (aChild->IsExpanded() &&
            nestedChildren && nestedChildren->entries() > 0)
            ConnectToChildren( aChild, theDC, theRegion );
        ....
    }
}
```

Advanced XVT-Power++

Slide 31

Managing Scrolling Ranges

- ❖ Three scrollbar management steps are needed
 - ♦ When the HierarchyBrowser changes size, adjust the large scrolling increments (page up/down) to reflect the new page size
 - ♦ When the font changes, adjust the small scrolling increment (line up/down) to reflect the new font
 - ♦ Each time a subtree is expanded or compressed, adjust the range of the scrollbars

Advanced XVT-Power++

Slide 32

Size Changes

- ❖ **Override the standard DoSize method to adjust the scrollbar increments after a size change**

```
void XHierarchyBrowser::DoSize( const CRect& theNewSize )
{
    CScroller::DoSize( theNewSize );
    ReSynch();
}

void XHierarchyBrowser::ReSynch( void )
{
    if (!itsRoot) return;
    EnlargeToFit( itsRoot->GetCurrentFrame().Inflate(2) );
    if (itIsHScrolling)
        SetHIncrements( itsIndentation,
                        GetInnerFrame().Width() - itsIndentation );
    if (itIsVScrolling)
        SetVIncrements( itsRoot->GetMinimumFrame().Height(),
                        GetInnerFrame().Height() -
                        itsRoot->GetMinimumFrame().Height() );
}
```

Advanced XVT-Power++

Slide 33

Font Changes

- ❖ **When the font changes**
 - ♦ The subtree class will recompute the average line height
 - ♦ Recompute a horizontal scrolling increment based on some average character widths

```
void XHierarchyBrowser::SetFont(
    const CFont& theFont,
    BOOLEAN isUpdate )
{
    CFont aFont = theFont;
    if (!aFont.IsMapped())
        aFont.Map(GetCWindow());
    CScroller::SetFont( aFont, isUpdate );
    itsIndentation = aFont.GetTextWidth( "Wo" );
    ReSynch();
}
```

Advanced XVT-Power++

Slide 34

Expansion Changes

- ❖ When a subtree is expanded or compressed, it needs to notify the hierarchy browser to change the scrolling ranges

- ♦ Could be implemented with ADP or with DoCommand

```
void XHierarchyBrowser::DoCommand( long theCommand, void* theData )
{
    switch (theCommand) {
        case TREEHeightChangeCmd: // TODO - Add width change too
        {
            UNITS aHeightChange = *(UNITS*)theData;
            CRect anOldVirtualFrame = GetVirtualFrame();
            UNITS aNewHeight = anOldVirtualFrame.Height() + aHeightChange;
            SetVirtualFrame( anOldVirtualFrame.Width(), aNewHeight );
        } break;
        ....
    }
}
```

Advanced XVT-Power++

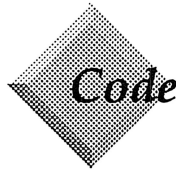
Slide 35

Consider Downward Chaining

- ❖ Which view methods and downward chaining methods make sense in the hierarchy browser?
 - ♦ SetDragging, SetSizing, DoSetDragging and DoSetSizing
 - Standard dragging and sizing behavior will interfere with the ability to select nodes or to implement drag and drop
 - ♦ SetGlue and DoSetGlue
 - Need to be able to define Glue for the hierarchy browser, but don't chain the notification downward
 - ♦ SetEnvironment, SetFont, DoSetEnvironment and DoSetFont
 - A shared environment, owned by the hierarchy object, can be used for background colors, fonts, and line styles; there is no need to downward chain
 - ♦ Other standard behaviors (visible, enabled, active) are fine as implemented by the framework

Advanced XVT-Power++

Slide 36



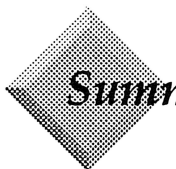
Code Walkthrough

- ❖ Review the code for XHierarchyBrowser



Advanced XVT-Power++

Slide 37



Summary: Writing a Subview

- ❖ Examine all possible subclassing options
- ❖ Design for both consumers and adaptors
- ❖ Get a clear understanding of frame concepts for the subview before beginning implementation
- ❖ Consider writing an initializer for consistency with the rest of the framework
- ❖ Implement Draw() and DoDraw(), then optimize
 - ♦ Remember to convert all coordinates to the same coordinate system
- ❖ Determine which downward chaining behaviors are appropriate for your class

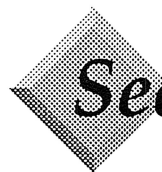


Advanced XVT-Power++

Slide 38

Section 3

Building a Subtree View



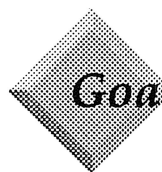
Section 3

Building a Subtree View



Advanced XVT-Power++

Slide 1



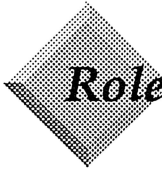
Goals of Section 3

- ❖ Design and implement a second custom subview class
- ❖ Examine how frame concepts apply to a custom view class
- ❖ Review coordinate translations
- ❖ Implement a Draw function to display text and images
- ❖ Review CFont properties and write code to respond to font changes



Advanced XVT-Power++

Slide 2



Roles and Responsibilities

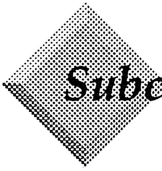
❖ XSubtree

- ♦ Draws the node information (text, images, etc)
- ♦ Manages frame concepts for the subtree
- ♦ Implements the subtree drop behavior
- ♦ Generates a notification when a node is selected, deselected or dropped
- ♦ Computes and maintains layout information based on the assigned font
- ♦ Others...?



Advanced XVT-Power++

Slide 3



Subclassing Options

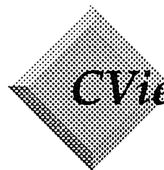
❖ Derivation options for the XSubtree class

- ♦ CView
- ♦ CText
- ♦ CPicture
- ♦ CSubview
- ♦ Others...?



Advanced XVT-Power++

Slide 4



CView Properties

❖ Useful Properties

- ♦ Draw function can be overridden to draw label and images
- ♦ Built-in frame concepts
- ♦ Generates commands when selected and double clicked

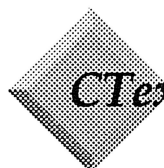
❖ Drawback

- ♦ Difficult to relate a parent node to its children
- ♦ Hierarchy class would need to know about each node



Advanced XVT-Power++

Slide 5



CText Properties

❖ Useful properties

- ♦ All CView properties, plus...
- ♦ Draws a label for each node
- ♦ Will respond to font changes

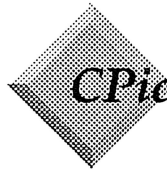
❖ Drawbacks

- ♦ All CView drawbacks, plus...
- ♦ Can't easily connect an image with the text
- ♦ Difficult to recompute tree heights on a font change



Advanced XVT-Power++

Slide 6



CPicture Properties

❖ Useful properties

- ♦ All CView properties, plus...
- ♦ It's a subview, so a parent node is connected to its children

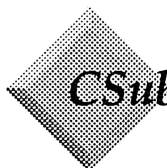
❖ Drawbacks

- ♦ Challenging to align text with image
- ♦ Adaptors may not want to draw bitmaps, only text
- ♦ Difficult to recompute tree heights on a font change



Advanced XVT-Power++

Slide 7



CSubview Properties

❖ Useful Properties

- ♦ Draw function can be overridden to draw label and images
- ♦ Generates commands on single and double click
- ♦ Built-in frame concepts
- ♦ Allows a subtree to easily own all its child nodes
- ♦ Hierarchy class only needs to know about a root node
- ♦ Downward chaining of DoDraw()
- ♦ Adaptors are free to add other types of views

❖ This is the best option for the XSubtree implementation



Advanced XVT-Power++

Slide 8

Deriving from CSubview

- ❖ For the basic XSubtree functionality, what needs to be implemented?
 - ♦ Constructors and destructor
 - ♦ Override Draw() to display the node information
 - ♦ Public API functions to allow consumers to query a subtree's properties
 - ♦ Frame management functions
 - ♦ Protected API functions to allow XHierarchyBrowser, a friend class, to modify a subtree's properties
 - ♦ Font handling functions
- ❖ Other features will be added later
 - ♦ Drag and drop
 - ♦ Printing

Advanced XVT-Power++

Slide 9

Constructors

- ❖ XSubtree is a private class instantiated by XHierarchyBrowser, so there are no constraints on the constructor

```
XSubtree::XSubtree( CSubview* theEnclosure,
                  const CPoint& theTopLeft,
                  XSubtree* theRoot,
                  const CStringRW& theLabel,
                  BOOLEAN isExpanded )
: CSubview( theEnclosure, CRect( theTopLeft, theTopLeft ) ),
  itsParent( theRoot ), itsIsExpanded( isExpanded ),
  itsExpandedImage( 0 ), itsCompressedImage( 0 ),
  itsLeafImage( 0 ), itsIsLeaf( TRUE ), itsIsSelected( FALSE )
{
  SetTitle( theLabel );
  itsLevel = theRoot ? theRoot->itsLevel + 1 : 0;
  Size( ComputeFrame( theTopLeft ) );
  SetCurrentFrame( itsFrame );
  SetCommand( SUBTREESelectedCmd );
  SetDoubleClickCommand( TOGGLEExpansionCmd );
}
```

Advanced XVT-Power++

Slide 10

Drawing the Node Information

- ❖ The Draw() method will draw a textual label, and possibly draw a selection box and a bitmap

- ♦ All coordinates must be converted to global coordinates

```
void XSubtree::Draw( const CRect& theClippingRegion )
{
    if (Prepare( theClippingRegion - GetClippedFrame() ) )
    {
        CDrawingContext aDC( this );
        if (itIsSelected)
            aDC.DrawRect( GetMinimumFrame().Globalize(GetEnclosure()));
        CPoint aLocation( 0, itsLeading+itsAscent );
        aLocation.Globalize( this );
        CImage* anImage = GetImage();
        if (anImage) {
            ... logic to draw the image ...
            aLocation.H( aLocation.H() + itsMarginWidth );
        }
        aDC.DrawText( itsTitle, aLocation );
    }
}
```

Advanced XVT-Power++

Slide 11

Drawing an Image

- ❖ CDrawingContext::DrawImage() lets you

- ♦ draw portions of a larger bitmap (sourceRect < bitmap size)
- ♦ scale bitmaps to smaller or larger sizes

```
void XSubtree::Draw( const CRect& theClippingRegion )
{
    ....
    CImage* anImage = GetImage();
    if (anImage) {
        CRect sourceRect( 0, 0,
            HLogical(ImageWidth), VLogical(ImageHeight) );
        CRect destRect = CRect( 0, 0, ImageWidth, ImageHeight )
            + CPoint(2, (itsLineHeight-ImageHeight)/2);
        destRect.Globalize( this );
        aDC.DrawImage( XVT_IMAGE(*anImage), sourceRect, destRect );
        aLocation.H( aLocation.H() + itsMarginWidth );
    }
    ....
}
```

Advanced XVT-Power++

Slide 12

Coordinate Conversions

- ❖ The drawing code ensures all coordinates passed to `CDrawingContext` are global

- ♦ Both `CRect` and `CPoint` provide several conversion functions

```
CPoint aLocation( 0, itsLeading+itsAscent );
aLocation.Globalize( this );

....

CRect destRect = CRect( 0, 0, ImageWidth, ImageHeight)
                  + CPoint(2, (itsLineHeight-ImageHeight)/2);
destRect.Globalize( this );
```

Advanced XVT-Power++

Slide 13

XSubtree Class - Consumer API

XSubtree
GetEntryPoint() GetExitPoint() IsExpanded() IsChildOf() GetCurrentFrame() GetMinimumFrame() GetImage() GetExpandedImage() GetCompressedImage() GetLeafImage()

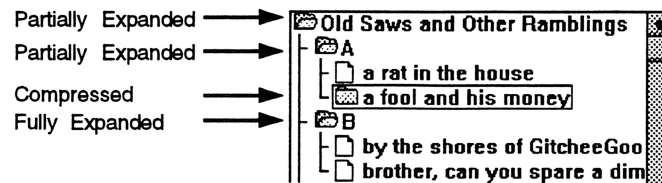
Advanced XVT-Power++

Slide 14

Subtree Frame Expansion

❖ A subtree can be in one of 3 states

- ♦ Compressed
- ♦ Partially expanded
- ♦ Fully expanded



Advanced XVT-Power++

Slide 15

Frame Concepts

❖ Have clearly delineated frame concepts

- ♦ GetFrame() returns the dimensions of the subtree at its full expansion
- ♦ GetCurrentFrame() returns the dimensions of the subtree at its current level of expansion
- ♦ GetMinimumFrame() returns the dimensions of the compressed subtree

❖ Implementing this requires maintaining an additional CRect data member, itsCurrentFrame

Advanced XVT-Power++

Slide 16

Frame Management

❖ When the expansion state of a subtree is toggled:

- ♦ The subtree changes height by the difference between itsCurrentFrame.Height() and GetMinimumFrame().Height()
- ♦ Each subtree that this subtree is a child of also changes height by the same amount
- ♦ Each subtree below this subtree needs to be moved by the height change

❖ The subtree class implements some protected functions to simplify this work

```
void ChildSizeChanged( UNITS theMaxHeightChange,  
                      UNITS theExpandedHeightChange,  
                      UNITS theWidthChange );  
void RepositionSiblings( const XSubtree* theNode,  
                        UNITS theHeightChange,  
                        UNITS theWidthChange );
```

Advanced XVT-Power++

Slide 17

XSubtree Class - Internal API

XSubtree
AddedChild() RemoveChild() ShowDetail() CalculateFrames() ComputeFrame() CurrentFrameOfChildren() SetCurrentFrame() SetExpandedImage() SetCompressedImage() SetLeafImage() ChildSizeChanged() RepositionSiblings() FontMetricsChanged()

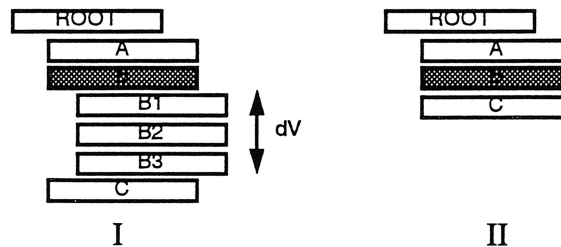
Advanced XVT-Power++

Slide 18

Expansion Changes

❖ Going from I to II

- ♦ Current height of B changes by $-dV$
- ♦ Current height of ROOT changes by $-dV$
- ♦ C must be moved by $-dV$



Advanced XVT-Power++

Slide 19

Frame Changes

```
void XSubtree::ChildSizeChanged( UNITS theMaxHeightChange,
                                UNITS theCurrentHeightChange,
                                UNITS theWidthChange )
{
    ... calculate newMax, newCurrent frames using input args ...
    DoSize( aNewMaxFrame );
    SetCurrentFrame( aNewCurrentFrame );
    // Propagate size changes up through the hierarchy
    if (itsParent) {
        theWidthChange = aNewMaxFrame.Right() -
            itsParent->GetFrame().Width();
        if (theWidthChange < 0) theWidthChange = 0;
        if (!itsParent->itIsExpanded) theCurrentHeightChange = 0;
        itsParent->ChildSizeChanged( theMaxHeightChange,
                                    theCurrentHeightChange,
                                    theWidthChange );
    }
}
```

Advanced XVT-Power++

Slide 20

Toggle Expansion

❖ The double click is used to change the expansion state

- ♦ Every subtree has its double command set to TOGGLEExpansionCmd

```
void XSubtree::DoCommand( long theCommand, void* theData )
{
    switch (theCommand) {
        case TOGGLEExpansionCmd:
            ShowDetail( !itIsExpanded );
            break;
        ....
    }
```

Advanced XVT-Power++

Slide 21

Coding the Expansion Change

```
void XSubtree::ShowDetail( BOOLEAN isExpanded )
{
    if (isExpanded != itIsExpanded) {
        ... calculate current height and width changes ...
        itIsExpanded = isExpanded;

        // adjust this subtree's frames by calculated deltas
        // and propagate the size change up thru the tree
        ChildSizeChanged( 0, aHeightChange, aWidthChange );

        if (itsParent)
            // notify this node's parent to move this node's siblings
            itsParent->RepositionSiblings(this, aHeightChange, 0, FALSE);
        ....
    }
}
```

Advanced XVT-Power++

Slide 22

Origins and Scrollers

- ❖ Because (and only because) the hierarchy browser is derived from CScroller, moving nodes can be accomplished with DoSetOrigin

- ♦ The origin concept is really only meaningful in virtual frames

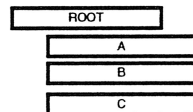
```
void XSubtree::RepositionSiblings( const XSubtree* theNode,
                                  UNITS theHeightChange, UNITS theWidthChange )
{
    .... locate theNode in the order of the children ....
    if (aChild == theNode) {
        // Move each sibling of theNode
        while ((aChild = (XSubtree*)nextChild()) != NULL) {
            aChild->DoSetOrigin( aDelta );
            aChild->itsCurrentFrame += aDelta;
        }
        ... propagate the change up to this node's parent ...
        ... redraw the area containing moved nodes ...
        ... notify the scroller of a scrolling range change ...
    }
}
```

Advanced XVT-Power++

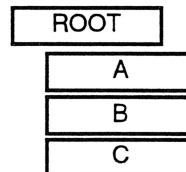
Slide 23

Font Changes

- ❖ When the user selects a new font
 - ♦ Recalculate the height and extent of each subtree
 - ♦ Change the scrolling range to fit the new current frame of the root node
 - ♦ Move each subtree down by the change in line height times the number of lines preceding this subtree



I



II

Advanced XVT-Power++

Slide 24

Overriding SetFont

- ❖ To respond to font changes, override the SetFont method

```
void XSubtree::SetFont(const CFont& theNewFont, BOOLEAN isUpdate)
{
    CFont aFont = theNewFont;
    if (!aFont.IsMapped()) {
        aFont.Map( GetCWindow() );
        FontMetricsChanged( aFont );
    }
    // propagate font change to all views sharing the environment
    CSubview::SetFont( aFont, isUpdate );

    // the root node will recalculate the extents of each subtree
    if (!itsParent)
        CalculateFrames();
}
```

Advanced XVT-Power++

Slide 25

Cumulative Moves

- ❖ How can you move each subtree by an amount that depends on how many objects precede it?

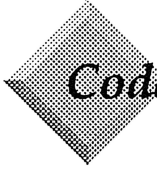
- ♦ DoSize() changes the size of a subview, and adjusts the origins of all the views inside it

```
void XSubtree::CalculateFrames( void )
{
    RWOIterator nextChild( *itsSubviews );
    XSubtree* aChild;
    CRect aFrame = GetMinimumFrame();
    CRect aCurrentFrame = aFrame;
    while ((aChild = (XSubtree*)nextChild()) != 0) {
        // depth-first calculation
        aChild->CalculateFrames();
        ... adjust Frame and CurrentFrame based on
        ... the new frames of all children
    }
    DoSize( aFrame );
    SetCurrentFrame( aCurrentFrame );
}
```

Important!

Advanced XVT-Power++

Slide 26



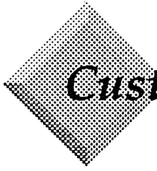
Code Walkthrough

- ❖ Review the code for XSubtree



Advanced XVT-Power++

Slide 27



Custom Subview Tips

- ❖ Sometimes the best class to derive from is the basic CView or CSubview
- ❖ CDrawingContext always uses global coordinates
- ❖ For the subtree class, a clear frame concept is key to successful implementation
- ❖ Remember - origins only have meaning inside virtual frames
- ❖ DoSize will move a subview and adjust all the origins of the nested views

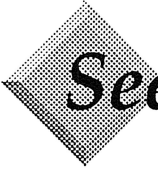


Advanced XVT-Power++

Slide 28

Section 4

Drag and Drop Techniques



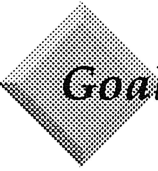
Section 4

Drag and Drop Techniques



Advanced XVT-Power++

Slide 1



Goals of Section 4

- ❖ Learn general concepts to customize the way in which mouse events are handled, without modifying the framework
- ❖ Understand a general drag-and-drop interface
 - ♦ Sources - where an object can be dragged from
 - ♦ Sinks - where a dragged object can be dropped
- ❖ Learn PTK functions for mouse event trapping and cursor management
- ❖ Implement drag-and-drop functionality in a custom subview class



Advanced XVT-Power++

Slide 2

Advanced Mouse Handling

- ❖ To add advanced features, you need to extend the way in which mouse events are handled without modifying the framework
- ❖ The default mouse handling behavior in Power++
 - ♦ Mouse event notifications are passed to a DoMouseXXX method on the window class
 - ♦ The default CWindow behavior is inherited from CSubview

```
void CSubview::DoMouseXXX(CPoint theSpot, short theButton,  
                          BOOLEAN isShiftKey, BOOLEAN isCtlKey)  
{  
    CView *anItem = FindEventTarget(theSpot)->FindHitView(theSpot);  
    theSpot = theLocation.Localize(anItem);  
    if (anItem->IsEnabled())  
        anItem->MouseXXX(theSpot, theButton, isShiftKey, isCtlKey);  
}
```

Advanced XVT-Power++

Slide 3

Mouse Managers and Handlers

- ❖ To change default mouse handling behavior, the XVT-Architect tutorial introduces two new classes
 - ♦ CMouseHandler provides the basic interface for implementing a specific technique of mouse event handling
 - ♦ CMouseManager allows a subview to have multiple types of mouse handlers
- ❖ Use these classes in your own applications
 - ♦ Instantiate a mouse manager in your derived window or subview
 - ♦ Derive your own custom mouse handler classes
- ❖ Important Note:
 - ♦ These are not productized classes, they are part of an example. The interface may change somewhat during productization.

Advanced XVT-Power++

Slide 4

Mouse Handler Interface

- ❖ CMouseHandler provides an interface for handling specific types of mouse notifications

```
virtual BOOLEAN DoDown(CPoint theLocation,
                      short theButton = 0,
                      BOOLEAN isShiftKey = FALSE,
                      BOOLEAN isControlKey = FALSE) = 0;
virtual BOOLEAN DoUp(CPoint theLocation,
                    short theButton = 0,
                    BOOLEAN isShiftKey = FALSE,
                    BOOLEAN isControlKey = FALSE) = 0;
virtual BOOLEAN DoDouble(CPoint theLocation,
                        short theButton = 0,
                        BOOLEAN isShiftKey = FALSE,
                        BOOLEAN isControlKey = FALSE) = 0;
virtual BOOLEAN DoMove(CPoint theLocation) = 0;
```

Advanced XVT-Power++

Slide 5

Mouse Manager Class

- ❖ The mouse manager maintains a linked list of mouse handlers and dispatches mouse events to each mouse handler in order

```
BOOLEAN CMouseManager::DoDown(const CWindow *theWindow,
                              CPoint theSpot,
                              short theButton,
                              BOOLEAN isShiftKey,
                              BOOLEAN isCtlKey)
{ // SIMPLIFIED Version
  RWGSlistIterator(CMouseHandler) nextHandler( itsMouseHandlers );
  BOOLEAN anEventConsumed = FALSE;
  while ( (aMouseHandler = nextHandler()) != NULL ) {
    anEventConsumed |= aMouseHandler->DoDown(
      theSpot, theButton, isShiftKey, isCtlKey);
  }
  return anEventConsumed;
}
```

Advanced XVT-Power++

Slide 6

Customized Mouse Handling

- ❖ To use these capabilities, follow these steps:
 - ♦ Derive your own mouse handler classes from CMouseHandler
 - ♦ Add a mouse manager to your derived subview or window class
 - ♦ Instantiate one or more mouse handlers and register each with the mouse manager
 - ♦ Override the DoMouse... methods on your derived window class
- Or -
- ♦ Override the Mouse... methods on your derived subview class to give the mouse manager first opportunity to handle incoming mouse events.
- ♦ Pass any unhandled events through the default handling.

Advanced XVT-Power++

Slide 7

Deriving a Mouse Handler

```
class XFlyByHelpHandler : public CMouseHandler
{
public:
    ... constructors/destructor ...

    virtual BOOLEAN DoMove( CPoint theLocation );
    virtual BOOLEAN DoUp( CPoint theLocation,
        short theButton = 0,
        BOOLEAN isShiftKey = FALSE,
        BOOLEAN isControlKey = FALSE ) {return FALSE;}
    virtual BOOLEAN DoDown(
        CPoint theLocation,
        short theButton = 0,
        BOOLEAN isShiftKey = FALSE,
        BOOLEAN isControlKey = FALSE ) {return FALSE;}
    virtual BOOLEAN DoDouble(
        CPoint theLocation,
        short theButton = 0,
        BOOLEAN isShiftKey = FALSE,
        BOOLEAN isControlKey = FALSE ) {return FALSE;}
    virtual BOOLEAN UsesGlobalCoords( void ) {return TRUE;}
```

Advanced XVT-Power++

Slide 8

Window with a Mouse Manager

```
class XMyWindow : public CWindow
{
public:
    ... standard methods ...
    virtual void DoMouseDouble(CPoint theLocation,
        short theButton = 0,
        BOOLEAN isShiftKey = FALSE,
        BOOLEAN isControlKey = FALSE);
    virtual void DoMouseDown( CPoint theLocation,
        short theButton = 0,
        BOOLEAN isShiftKey = FALSE,
        BOOLEAN isControlKey = FALSE);
    virtual void DoMouseUp( CPoint theLocation,
        short theButton = 0,
        BOOLEAN isShiftKey = FALSE,
        BOOLEAN isControlKey = FALSE);
    virtual void DoMouseMove( CPoint theLocation,
        short theButton = 0,
        BOOLEAN isShiftKey = FALSE,
        BOOLEAN isControlKey = FALSE);

protected:
    CMouseManager itsMouseManager;
```

Advanced XVT-Power++

Slide 9

Subview with a Mouse Manager

```
class XMySubview : public CSubview
{
public:
    ... standard methods ...
    virtual void MouseDouble(CPoint theLocation,
        short theButton = 0,
        BOOLEAN isShiftKey = FALSE,
        BOOLEAN isControlKey = FALSE);
    virtual void MouseDown( CPoint theLocation,
        short theButton = 0,
        BOOLEAN isShiftKey = FALSE,
        BOOLEAN isControlKey = FALSE);
    virtual void MouseUp( CPoint theLocation,
        short theButton = 0,
        BOOLEAN isShiftKey = FALSE,
        BOOLEAN isControlKey = FALSE);
    virtual void MouseMove( CPoint theLocation,
        short theButton = 0,
        BOOLEAN isShiftKey = FALSE,
        BOOLEAN isControlKey = FALSE);

protected:
    CMouseManager itsMouseManager;
```

Advanced XVT-Power++

Slide 10

Register Mouse Handler(s)

```
XMyWindow::XMyWindow(CDocument *theDocument,
                    const CRect& theRegion,
                    const CStringRW& theTitle,
                    long theAttributes,
                    WIN_TYPE theWindowType,
                    int theMenu)
: CWindow(theDocument, theRegion, theTitle,
          theAttributes, theWindowType, theMenu)
{
    XFlyByHelpHandler* aHandler1 = new XFlyByHelpHandler();
    XPopupMenuHandler* aHandler2 = new XPopupMenuHandler();
    itsMouseManager.RegisterMouseHandler( aHandler1 );
    itsMouseManager.RegisterMouseHandler( aHandler2 );
    ...
}
```

Advanced XVT-Power++

Slide 11

Dispatch Events to the Manager

```
void XMyWindow::DoMouseDown(
    CPoint theLocation, short theButton,
    BOOLEAN isShiftKey, BOOLEAN isCtlKey)
{
    if (!itsMouseManager.DoDown( this, theLocation,
                                isShiftKey, isCtlKey ))
        CWindow::DoMouseDown(theLocation,theButton,isShiftKey,isCtlKey);
}

... handle DoMouseUp and DoMouseDouble like DoMouseDown ...

void XMyWindow::DoMouseMove(
    CPoint theLocation, short theButton,
    BOOLEAN isShiftKey, BOOLEAN isCtlKey)
{
    if (!itsMouseManager.DoMove( this, theLocation ))
        CWindow::DoMouseMove(theLocation,theButton,isShiftKey,isCtlKey);
}
```

Advanced XVT-Power++

Slide 12

Drag and Drop

- ❖ Drag and drop techniques are a specific type of mouse event handling
- ❖ Two key concepts:
 - ♦ A specialized mouse handler is needed to determine when an object is being dragged
 - ♦ An additional abstraction is needed to identify places where an object can be dropped
- ❖ Other considerations
 - ♦ Once dragging starts, the subview needs to see all mouse events until the object is ~~dragged~~ dropped.
 - ♦ Typically the cursor changes to indicate an object is being dragged or copied, and may change again if the mouse pointer moves over an invalid drop site

Advanced XVT-Power++

Slide 13

Drag-and-Drop Terminology

- ❖ A drag *sink* is an object or location that a dragged object can be dropped on
 - ♦ Sinks register with Sources as valid drop sites for that source
- ❖ A drag *source* is a specialized mouse handler. It:
 - ♦ maintains a list of sinks
 - ♦ notifies a sink when a dragged object enters, moves over or leaves the sink region
 - ♦ manages mouse trapping and cursor shapes while an object is dragged and restores them when it is dropped
- ❖ You should derive a drag source that decides when an object is being dragged
 - ♦ The tutorial derives a TDragWindow class

Advanced XVT-Power++

Slide 14

CDragSink

- ❖ **CDragSink is an abstract class defining the standard sink interface**
 - ♦ A sink is a place where a dragged object can be dropped
- ❖ **A drag sink is notified by a drag source when**
 - ♦ an object is dragged into the sink region
`virtual void DoEnter( long theDragData );`
 - ♦ an object is being dragged within the sink region
`virtual void DoDrag( CPoint theLocation, long theDragData );`
 - ♦ an object is dragged out of the sink region
`virtual void DoLeave( long theDragData );`
 - ♦ an object is dropped on the sink region
`virtual void DoDrop( CPoint theLocation, long theDragData );`

 Advanced XVT-Power++

Slide 15

Specialized Sinks

- ❖ **The XVT-Architect tutorial provides two derivations of the abstract CDragSink**
 - ♦ CViewSink allows you to designate any view as a drop site
 - ♦ CRectSink allows you to designate an arbitrary rectangular region as a drop site
- ❖ **Use these directly, or use them as a model for your own drop site classes**

 Advanced XVT-Power++

Slide 16

Mouse Trapping and Cursors

- ❖ **The PTK provides pass-through functionality that is used to implement drag-and-drop**

- ♦ **Logically trap the mouse to a window, so that all mouse events are delivered to that window**

```
void xvt_win_trap_pointer( WINDOW w );
```

- ♦ **Resume standard delivery of mouse events**

```
void xvt_win_release_pointer( void );
```

- ♦ **Cursormangement functions**

```
CURSOR xvt_win_get_cursor( WINDOW w );
```

```
void xvt_win_set_cursor( WINDOW w, CURSOR c );
```

- ❖ **CDragSource uses these functions when dragging begins and when an object is dropped**

- ♦ **Derived Sink classes may do additional cursor management**

Advanced XVT-Power++

Slide 17

Drag-and-Drop Cookbook

- ❖ **Add a mouse manager to your derived window or subview class, and override the default mouse methods**
- ❖ **Derive from CDragSource to determine when an object is being dragged, and register this mouse handler with the mouse manager**
- ❖ **Derive from CDragSink to implement IsInSink(), and to define enter, leave and drop behavior**
- ❖ **Instantiate one or more of your drag site objects and register them with the drag source**

Advanced XVT-Power++

Slide 18

Subview With a Mouse Manager

- ❖ **Override the standard MouseXXX methods on a subview class to plug in the mouse handlers**

```
void XHierarchyBrowser::MouseDown( CPoint theSpot,
                                   short theBtn,
                                   BOOLEAN isShiftKey,
                                   BOOLEAN isCtlKey )
{
    if (!itsMouseManager.DoDown( GetCWindow(),
                                  theSpot.Globalize(this), theBtn,
                                  isShiftKey, isCtlKey )) {
        // if not a drag-begin, pass to standard command generation
        CView *aView = FindSubview(theLocation);
        if (aView)
            aView->DoMouseDown(theSpot, theBtn, isShiftKey, isCtlKey);
        else
            CSubview::MouseDown(theSpot, theBtn, isShiftKey, isCtlKey);
    }
}
```

Advanced XVT-Power++

Slide 19

Derive from CDragSource

- ❖ **Derive a custom mouse handler to identify the start of a drag. CDragSource functionality handles the behavior once a drag has begun**

```
class XDragStarter : public CDragSource {
public:
    XDragStarter(CWindow* theWindow,
                 CURSOR theMoveCursor, CURSOR theCopyCursor);
    .... copy ctor/assignment op/destructor ...

    // override Mouse methods to identify start of drag
    virtual BOOLEAN DoUp(...);
    ... DoDown/DoDouble/DoMove ...

protected:
    BOOLEAN itWantsToDrag;
    CURSOR itsDragCursor;
    CURSOR itsMoveCursor;
    CURSOR itsCopyCursor;
};
```

Advanced XVT-Power++

Slide 20

Finding an Object to Drag

- ❖ To identify the start of a drag, the handler must search for a draggable object on each mouse down

- ♦ In this code, a sink is both a place where an object can drop and an object that can be dragged

```
BOOLEAN XDragStarter::DoDown( CPoint theLocation,
    short /*theBtn*/, BOOLEAN /*isShift*/, BOOLEAN isCtlKey )
{ // find a draggable object beneath the mouse click
    itWantsToDrag = FALSE;
    RWGSListIterator(CDragSink) nextSink( itsSinks );
    CDragSink *aSink;
    while ((aSink = nextSink()) != 0)
        if ( aSink->IsInSink( theLocation ) ) {
            itsLastLocation = theLocation;
            itsLastSink = aSink; itWantsToDrag = TRUE;
            itsDragCursor = isCtlKey ? itsCopyCursor : itsMoveCursor;
            break;
        }
    return FALSE;
}
```

Advanced XVT-Power++

Slide 21

A Drag Begins by Calling DoDrag

- ❖ Examine mouse moves if not already dragging and the DoDown found a draggable object

```
BOOLEAN XDragStarter::DoMove( CPoint theLocation )
{
    if (itIsDragging) return CDragSource::DoMove( theLocation );
    else if (!itWantsToDrag) return FALSE;

    CPoint aDelta = theLocation - itsLastLocation;
    if ( abs(aDelta.H() ) < 1 && abs(aDelta.V() ) < 1 )
        return TRUE;

    itsLastLocation = theLocation;
    DoDrag( (long)itsLastSink, itsTrappedWindow, itsDragCursor );
    itWantsToDrag = FALSE;
    return TRUE;
}
```

Advanced XVT-Power++

Slide 22

Drag Data

- ❖ When a drag begins, pass in data to be delivered to the sink on which the object is dropped

```
virtual void DoDrag( long theDragData,  
                    CWindow *theTrapWindow,  
                    CURSOR theDragCursor );
```

- ❖ In this example, the code passes a pointer to the object that is being dragged

```
DoDrag( (long)itsLastSink, itsTrappedWindow, itsDragCursor );
```

- ❖ The drag source then passes this data on each of the enter, drag, leave and drop callbacks.

Advanced XVT-Power++

Slide 23

Drag and Drop Notifications

- ❖ After DoDrag has been invoked, CDragSource takes over to implement the sink enter-drag-leave notifications and the drop behavior

```
BOOLEAN CDragSource::DoMove( CPoint theSpot )  
{  
    .... search for the first sink containing theSpot ....  
    if ( aNewSink ) {  
        // Entering new sink, call DoLeave on old and DoEnter on new  
        if ( aNewSink != itsLastSink ) {  
            if ( itsLastSink )  
                itsLastSink->DoLeave( itsDragData );  
            aNewSink->DoEnter( itsDragData );  
        }  
        aNewSink->DoDrag( theLocation, itsDragData );  
    }  
    // if no new sink is found, call DoLeave on old sink  
    else if ( itsLastSink )  
        itsLastSink->DoLeave( itsDragData );  
    ....  
}
```

Advanced XVT-Power++

Slide 24

Derive a Drag Sink

❖ To implement a class representing a place or object on which a dragged object can be dropped:

- ♦ Derive a concrete sink class from the abstract CDragSink
- ♦ Override the 5 pure virtual functions provided by CDragSink: IsInSink, DoEnter, DoLeave, DoDrag and DoDrop

```
class CDragSink
{
public:
    virtual ~CDragSink();
    virtual BOOLEAN IsInSink( CPoint theLocation ) = 0;
    virtual void DoDrop( CPoint theLocation, long theDragData ) = 0;
    virtual void DoEnter( long theDragData ) = 0;
    virtual void DoLeave( long theDragData ) = 0;
    virtual void DoDrag( CPoint theLocation, long theDragData ) = 0;
protected:
    CDragSink( void );
    ....
};
```

Advanced XVT-Power++

Slide 25

IsInSink

❖ The IsInSink() method determines if the mouse cursor is over a drop site

```
BOOLEAN XSubTreeSink::IsInSink( CPoint theLocation )
{
    return (
        itsNode->IsVisible() &&
        itsNode->GetMinimumFrame().IsPointInRect(
            theLocation.GetTranslated( CObjectRWC::G->GetTaskWin(),
            itsNode->GetEnclosure() )
        )
    );
}
```

Advanced XVT-Power++

Slide 26

DoEnter

- ❖ The DoEnter() method defines behavior when the mouse cursor first moves over each drop site

```
void XSubTreeSink::DoEnter( long theDragData )
{
    // hilite the node if a valid drop site
    XSubTreeSink* aDraggedSink = (XSubTreeSink*)theDragData;
    if ( itsNode != aDraggedSink->itsNode &&
        !itsNode->IsChildOf( aDraggedSink->itsNode ) ) {
        CDrawingContext aDC( itsNode );
        CEnvironment anEnv = aDC.GetEnv();
        anEnv.SetDrawingMode( M_XOR );
        aDC.SetEnv( anEnv );
        aDC.DrawRect( itsNode->GetClippedFrame() -
                      itsNode->GetMinimumFrame().Globalize(
                          itsNode->GetEnclosure() ) );
    } else { // change to a no drop cursor if can't drop here
        WINDOW w = itsNode->GetCWindow()->GetXVTWindow();
        itsSavedCursor = xvt_win_get_cursor( w );
        xvt_win_set_cursor( w, CURSOR_NODROP );
    }
}
```

Advanced XVT-Power++

Slide 27

DoLeave

- ❖ The DoLeave() method defines behavior when the mouse cursor moves off of each drop site

```
void XSubTreeSink::DoLeave( long theDragData )
{
    // Get rid of hiliting on node being abandoned
    XSubTreeSink* aDraggedSink = (XSubTreeSink*)theDragData;
    if ( itsNode != aDraggedSink->itsNode &&
        !itsNode->IsChildOf( aDraggedSink->itsNode ) ) {
        CDrawingContext aDC( itsNode );
        CEnvironment anEnv = aDC.GetEnv();
        anEnv.SetDrawingMode( M_XOR );
        aDC.SetEnv( anEnv );
        aDC.DrawRect( itsNode->GetClippedFrame() -
                      itsNode->GetMinimumFrame().Globalize(
                          itsNode->GetEnclosure() ) );
    } else { // restore original cursor
        xvt_win_set_cursor( itsNode->GetCWindow()->GetXVTWindow(),
                          itsSavedCursor );
        itsSavedCursor = 0;
    }
}
```

Advanced XVT-Power++

Slide 28

DoDrag and DoDrop

- ❖ The DoDrop() method defines behavior when the object is dropped on a drop site; DoDrag() defines the behavior when the cursor is over a drop site

```
void XSubTreeSink::DoDrop( CPoint theLocation, long theDragData )
{
    if ( itsDropCommand != NULLcmd ) {
        CDragDropInfo aDragDropInfo( theLocation, theDragData );
        itsNode->DoCommand( itsDropCommand, &aDragDropInfo );
    }
}

void XSubTreeSink::DoDrag( CPoint theLocation, long theDragData )
{
    if ( itsDragCommand != NULLcmd ) {
        CDragDropInfo aDragDropInfo( theLocation, theDragData );
        itsNode->DoCommand( itsDragCommand, &aDragDropInfo );
    }
}
```

Advanced XVT-Power++

Slide 29

Create the Drag Sinks

- ❖ Once a custom drag sink class is defined, create a drag sink each time a new subtree node is created

```
XSubtree::XSubtree( CSubview* theEnclosure,
                   const CPoint& theTopLeft,
                   XSubtree* theRoot,
                   const CStringRW& theLabel,
                   BOOLEAN isExpanded )
: CSubview( theEnclosure, CRect( theTopLeft, theTopLeft ) ),
  ... initialize internal data members ...
{
    ....
    itsSinkArea = new XSubtreeSink( this, TREEDroppedCmd );
    DoCommand( REGISTERSinkCmd, itsSinkArea );
}
```

Advanced XVT-Power++

Slide 30

Register the Sinks

- ❖ The hierarchy browser object maintains the list of drag sinks

```
void XHierarchyBrowser::DoCommand(long theCommand, void* theData)
{
    switch (theCommand) {
        case REGISTERSinkCmd:
            if (itAllowsDragging)
                itsDragStarter->RegisterSink((CDragSink*)theData, TRUE);
            break;
        ...
        default:
            CScroller::DoCommand( theCommand, theData );
            break;
    }
}
```

Advanced XVT-Power++

Slide 31

Respond to the Drop Notification

- ❖ The custom sink class will notify a subtree with a command when another subtree is dropped on it

```
void XSubtree::DoCommand( long theCommand, void* theData )
{
    switch (theCommand) {
        case TREEDroppedCmd:
        {
            CDragDropInfo *aDragDropInfo = (CDragDropInfo*)theData;
            XSubtree* theMovedTree = aDragDropInfo->GetSubtree();
            // Make sure this drop was legal
            if (theMovedTree &&
                theMovedTree != this && // node dropped on itself
                !IsChildOf( theMovedTree ) && // parent dropped on a child
                theMovedTree->itsParent ) // can't move the root node
                MoveSubtree( theMovedTree );
        } break;
        ....
    }
```

Advanced XVT-Power++

Slide 32

Moving a Subtree

- ❖ When an object is dragged and dropped, several rearrangements are needed

- ♦ Remember to generate a notification command to the application programmer

```
void XSubtree::MoveSubtree( XSubtree* theMovedTree )
{
    // remove the node from the hierarchy at its old location
    // Move all the siblings of this node down to make room
    // connect the dropped node into the hierarchy
    // Recalculate the frames of this node with its new child
    // redraw the union of the old location and the new location
    // Notify app programmer of move, with appropriate data
    CSubview::DoCommand( TREEMovedCmd, theMovedTree );
}
```

Advanced XVT-Power++

Slide 33

Code Walkthrough

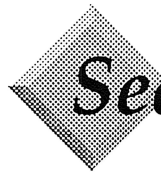
- ❖ Add drag and drop capabilities to the hierarchy browser

Advanced XVT-Power++

Slide 34

Section 5

Rogue Wave Tools.h++ Classes

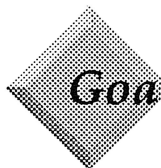


Section 5

Rogue Wave Tools.h++ Classes

 Advanced XVT-Power++

Slide 1

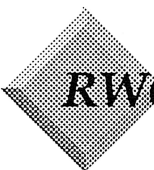


Goals of Section 5

- ❖ Understand the RWCollectable and RWCollection interfaces
- ❖ Use Iterators to traverse a collection
- ❖ Learn to use regular expression, tokenizer and substring classes
- ❖ Understand locales

 Advanced XVT-Power++

Slide 2



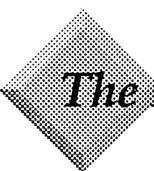
RWCollection & RWCollectable

- ❖ **RWCollection is the base class for the persistable Tools.h++ data structure classes**
 - ♦ The objects stored in any RWCollection must be derived from RWCollectable
- ❖ **RWCollectable defines a standard interface so a collection can sort, locate and persist the items in a collection**
 - ♦ One constraint - each RWCollectable must have a constructor with no required arguments



Advanced XVT-Power++

Slide 3



The RWCollectable Interface

- ❖ **To place collectables in a self-sorting collection, or if you need to search the collection for specific objects, your derived collectable class should override comparison and/or hashing methods**
- ❖ **Comparison interface**
 - ♦ Comparison for exact match

```
virtual RWBoolean isEqual(const RWCollectable*) const;
```
 - ♦ Comparison for greater than (positive return value), less than (negative return value) or equal (zero return value)

```
virtual int compareTo(const RWCollectable*) const;
```
- ❖ **Hashing interface for hashing-based collections**

```
virtual unsigned hash() const;
```



Advanced XVT-Power++

Slide 4

RWCollectable Interface (cont.)

- ❖ To persist a collection, the collectables must implement the persistence methods

```
virtual RWorkspace      binaryStoreSize() const;
virtual void            saveGuts(RWFile&) const;
virtual void            saveGuts(RWVostream&) const;
virtual void            restoreGuts(RWFile&);
virtual void            restoreGuts(RWVistream&);
```

Advanced XVT-Power++

Slide 5

RWCollectable Example

```
class XEmployee : public RWCollectable
{
    RWDECLARE_COLLECTABLE( XEmployee );
public:
    ... constructors and destructors ...
    virtual RWBoolean isEqual( const RWCollectable* c ) const;
    virtual int compareTo( const RWCollectable* c ) const;
    ... methods to get and set attributes ...
protected:
    long      itsID;
    RWDate    itsHireDate;
    CStringRW itsLastName;
    CStringRW itsFirstName;
    CStringRW itsJobTitle;
    ... additional data members ...
};
#define XEmployeeId 31200
```

Advanced XVT-Power++

Slide 6

RWCollectable Example (cont.)

```
RWDEFINE_COLLECTABLE( XEmployee, XEmployeeID );

RWBoolean XEmployee::isEqual( const RWCollectable* c ) const
{
    return ((XEmployee*)c)->GetEmployeeID() == itsID;
}

int XEmployee::compareTo( const RWCollectable* c ) const
{
    // default comparison to sort by id
    return
        isEqual( c ) ? 0 :
        ((XEmployee*)c)->GetEmployeeID() < itsID ? -1 : 1;
}
```

Advanced XVT-Power++

Slide 7

RWCollections

❖ The RWCollection classes can be organized into three major categories

♦ No concept of ordering in the collection

RWHashTable RWSet RWIdentityDictionary
RWIdentitySet RWBag RWHashDictionary

♦ Collection maintains an order as specified by the programmer

RWSListCollectables RWSListCollectablesQueue
RWDlistCollectables RWSListCollectablesStack
RWOordered

♦ Collection self-sorts when a new item is added

RWSortedVector RWBTreeDictionary
RWBTree RWBTreeOnDisk
RWBinaryTree

Advanced XVT-Power++

Slide 8

Basic RWCollection Methods

❖ **Determining the size of a collection**

```
virtual size_t      entries(void) const;  
virtual RWBoolean   isEmpty(void) const;
```

❖ **Adding to a collection**

```
virtual RWCollectable* insert(RWCollectable*);
```

❖ **Removing an item from a collection**

```
virtual RWCollectable* remove(const RWCollectable*);  
virtual void           removeAndDestroy(const RWCollectable*);
```

❖ **Removing everything from a collection**

```
virtual void          clear(void);  
virtual void          clearAndDestroy(void);
```

❖ **Checking for an item in a collection**

```
virtual RWBoolean     contains(const RWCollectable*) const;  
virtual RWCollectable* find(const RWCollectable*);  
virtual size_t         occurrencesOf(const RWCollectable*) const;
```

Advanced XVT-Power++

Slide 9

RWSequencable Methods

❖ **Collections that are ordered by the programmer provide several methods for managing the order**

◆ **Adding an element at a specific location**

```
virtual RWCollectable* append(RWCollectable*);  
virtual RWCollectable* insertAt(size_t, RWCollectable*);  
virtual RWCollectable* prepend(RWCollectable*);
```

◆ **Retrieving an element at a specific location**

```
virtual RWCollectable*& at(size_t);  
virtual const RWCollectable* at(size_t) const;  
virtual RWCollectable* first(void) const;  
virtual RWCollectable* last(void) const;
```

◆ **Determine the location of an element**

```
virtual size_t         index(RWCollectable*) const;
```

Advanced XVT-Power++

Slide 10

Traversing a Collection

- ❖ Every collection has an `apply()` function to traverse the collection and call the same function on each element of the collection

```
virtual void apply(RWapplyCollectable, void*);  
typedef void (*RWapplyCollectable)( RWCollectable*, void*);
```

- ❖ Also, every collection has a `select()` function for creating a new collection containing the members of the original collection that match some criteria

- ♦ You are responsible for delete'ing the returned collection

```
virtual RWCollection* select(RWtestCollectable, void*);  
typedef RWBoolean (*RWtestCollectable)(const RWCollectable*,  
                                       const void*);
```

Advanced XVT-Power++

Slide 11

Traversing a Collection (cont.)

- ❖ The other technique for traversing a collection is to instantiate an iterator object for the collection

- ♦ Each collection class has an associated iterator class

```
CSubview::DoDisable( void )  
{  
    Disable();  
    RWOrderedIterator nextView( *itsSubviews );  
    CView* aView;  
    while ((aView = (CView*)nextView()) != NULL)  
        aView->DoDisable();  
}
```

Advanced XVT-Power++

Slide 12

String Manipulation

❖ Tools.h++ provides several classes useful for string manipulation

- ♦ RWCTokenizer - breaks a string into tokens separated by user-defined delimiters
- ♦ RWCTRegexp - implements regular expression pattern matching
- ♦ RWCTSubString - enables manipulation of a subsection of an RWCTString or CStringRW by defining a starting position and an extent
- ♦ RWCTLocale and RWCTLocaleSnapshot - provides an interface for formatting numbers, times, dates and currency values

Advanced XVT-Power++

Slide 13

Tokenizer

❖ The tokenizer class breaks a string into tokens separated by delimiters

```
class RWCTokenizer
{
public:
    RWCTokenizer(const RWCTString& theString);
    // Advance to next token, delimited by theDelimiters:
    RWCTSubString operator()(const char* theDelimiters);
    RWCTSubString operator()();
private:
    ...
};
```

Advanced XVT-Power++

Slide 14

Using RWCTokenizer

```
RWCString aPath( " ./usr/bin;/usr/5bin;" );
RWCTokenizer nextDir( aPath );
RWCString aDirName;

while(!(aDirName = nextDir()).isNull())
    // use the directory name to check for a file
    ...
```

Advanced XVT-Power++

Slide 15

Regular Expressions

```
class RWExport RWRegexp
{
public:
    enum statVal {OK=0, ILLEGAL, NOMEM, TOOLONG};
    RWRegexp(const char*);
    RWRegexp(const RWCString&);
    RWRegexp(const RWRegexp&);
    ~RWRegexp();

    RWRegexp& operator=(const RWRegexp&);
    RWRegexp& operator=(const RWCString&); // Recompile pattern
    RWRegexp& operator=(const char*);    // Recompile pattern
    size_t index(const RWCString& str,
                  size_t* len,
                  size_t start=0) const;
    statVal status(); // Return & clear status
private:
    ...
};
```

Advanced XVT-Power++

Slide 16

Regular Expression Syntax

❖ Regular expression syntax matches the syntax of standard Unix tools `ed`, `sed` and the `grep` family

<code>c</code>	any non-special character matches itself
<code>\c</code>	turns off any special meaning of the character
<code>.</code>	matches any character except newline
<code>[...]</code>	matches any character in ...
<code>[^...]</code>	matches any character not in ...
<code>r*</code>	matches zero or more occurrences of <code>r</code>
<code>r+</code>	matches one or more occurrences of <code>r</code>
<code>r?</code>	matches zero or one occurrences of <code>r</code>
<code>^r</code>	matches a line that begins with <code>r</code>
<code>r\$</code>	matches a line that ends with <code>r</code>

Advanced XVT-Power++

Slide 17

Using Regular Expressions

```
BOOLEAN XEmployee::NameMatches(const XEmployee* thePattern) const
{
    RWRegexp    aFNameRegExp(".*"), aMNameRegExp(".*"),
                aLNameRegExp(".*");

    if (thePattern->GetFirstName() != "")
        aFNameRegExp = thePattern->GetFirstName();
    if (thePattern->GetMiddleName() != "")
        aMNameRegExp = thePattern->GetMiddleName();
    if (thePattern->GetLastName() != "")
        aLNameRegExp = thePattern->GetLastName();

    return      aFNameRegExp.status() == RWRegexp::OK &&
               aMNameRegExp.status() == RWRegexp::OK &&
               aLNameRegExp.status() == RWRegexp::OK &&
               itsFirstName.index( aFNameRegExp ) != RW_NPOS &&
               itsMiddleName.index( aMNameRegExp ) != RW_NPOS &&
               itsLastName.index( aLNameRegExp ) != RW_NPOS ;
}
```

Advanced XVT-Power++

Slide 18

Substrings

```
class RWExport RWSubString
{
public:
    RWSubString(const RWSubString& SP);
    RWSubString& operator=(const char*);
    RWSubString& operator=(const RWCString&);
    char& operator()(size_t i);
    char& operator[](size_t i);
    char operator()(size_t i) const;
    char operator[](size_t i) const;
    const char* data() const;
    size_t length() const;
    size_t start() const;
    void toLower(); // Convert to lower-case
    void toUpper(); // Convert to upper-case
    RWBoolean isNull() const;
    int operator!() const;
protected:
    ....
};
```

Advanced XVT-Power++

Slide 19

Using Substrings

❖ Substrings are created by operators or functions of the RWCString class

◆ You can't instantiate a RWSubString

```
RWRegexp aPattern( "NAME" );
RWCString aTestString = "Welcome to NAME's Wild Ride";
aTestString( aPattern ) = "Mr. Toad";
xvt_dm_post_note( "%s", aTestString.data() );
```

◆ This will display a dialog with the message

Welcome to Mr. Toad's Wild Ride

Advanced XVT-Power++

Slide 20

Locales

- ❖ **The RWLocale class provides an interface for converting between strings and basic data types**

```
virtual RWCString asString(long) const;
virtual RWCString asString(unsigned long) const;
virtual RWCString asString(
    double f, int precision = 6, RWBoolean showpoint = 0) const;
enum CurrSymbol { NONE, LOCAL, INTL };
virtual RWCString moneyAsString(double, CurrSymbol=LOCAL) const;

virtual RWBoolean stringToNum (const RWCString&, double*) const;
virtual RWBoolean stringToNum (const RWCString&, long*) const;
virtual RWBoolean stringToDate (const RWCString&, struct tm*) const;
virtual RWBoolean stringToTime (const RWCString&, struct tm*) const;
virtual RWBoolean stringToMoney(const RWCString&, double*,
                                CurrSymbol=LOCAL) const;

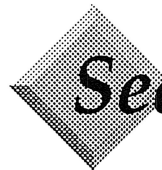
static const RWLocale& global();
```

Advanced XVT-Power++

Slide 21

Section 6

Advanced Command Handling



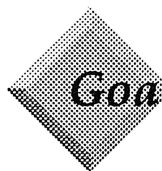
Section 6

Advanced Command Handling



Advanced XVT-Power++

Slide 1



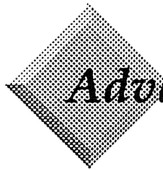
Goals of Section 6

- ❖ Learn advanced techniques for structuring command handling code
 - ♦ Run-time modification of command handling
 - ♦ Commands that require a reply
 - ♦ Broadcasting



Advanced XVT-Power++

Slide 2



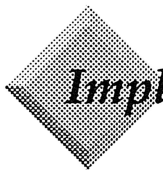
Advanced Command Handling

- ❖ A switch statement may not be the optimum code management technique for handling large numbers of commands
- ❖ A lookup table that maps commands to handler functions has several benefits
 - ◆ Ability to change command handling at runtime
 - ◆ More manageable code structure
 - ◆ Personal preference. You may simply prefer a callback style to a switch statement style of programming



Advanced XVT-Power++

Slide 3



Implementing a Lookup Table

- ❖ The lookup table collects pairs of items - a command and a handler function
- ❖ Choosing a collection class - the thought process
 - ◆ Order inside the lookup table is not important
 - ◆ Optimize for lookup speed
 - ◆ Insertion and deletion will be done infrequently, speed not as important
 - ◆ Duplicates are not allowed
- ❖ Other considerations
 - ◆ Maximum size of collection - 64 commands?
 - ◆ Store as key and value, or as a single object encapsulating both the command and the handler?



Advanced XVT-Power++

Slide 4

A Command Dispatcher

- ❖ Design a command dispatcher class to store a command and a callback function pointer
 - ♦ Implement using a hash dictionary. The lookup key will be the command, the value will be the callback function pointer
- ❖ Declare a typedef for the callback function to keep the code readable
 - ♦ The callback function is a member function on an object derived from CView. The function has a void return value, and takes two arguments, a long and a void* (the arguments passed to DoCommand)

```
typedef void (CView::*CMDCallback)(long, void *);
```

Advanced XVT-Power++

Slide 5

Command Dispatcher Interface

```
class XCmdDispatcher
{
public:
    // XCmdDispatcher callback function signature:
    typedef void (CView::*CMDCallback)(long, void *);

    XCmdDispatcher() {}
    virtual ~XCmdDispatcher()
    { itsCallbackTable.clearAndDestroy(); }

    void InsertCmd(CMDCallback itsCallback,
                  long theCommand,
                  BOOLEAN theNewReplacesExistingCallback = TRUE);

    BOOLEAN DispatchCmd(CView* theObject,
                       long theCommand,
                       void* theData) const;
protected:
    RHashDictionary itsCallbackTable;
};
```

Advanced XVT-Power++

Slide 6

Using an RWCollection

- ❖ In order to enter commands and callbacks into the lookup table (RWHashDictionary), the classes need to be derived from RWCollectable
 - ♦ Use CLongRWC to store the commands as keys in the lookup table
 - ♦ Derive a wrapper class from RWCollectable to store callback functions in the lookup table

Advanced XVT-Power++

Slide 7

RWCollectable Callback Function

```
class XCMDCallbackRWC : public RWCollectable
{
    RWDECLARE_COLLECTABLE(XCMDCallbackRWC)
public:
    XCMDCallbackRWC(XCmdDispatcher::CMDCallback theCallback)
        : itsCallback(theCallback) {}
    virtual ~XCMDCallbackRWC() {}

    XCmdDispatcher::CMDCallback value(void) const
    { return itsCallback; }

protected:
    XCmdDispatcher::CMDCallback itsCallback;
};

#define XCMDCallbackRWCID 40001
RWDEFINE_COLLECTABLE(XCMDCallbackRWC, XCMDCallbackRWCID)
```

Advanced XVT-Power++

Slide 8

Dispatcher Implementation

- ❖ The `InsertCmd()` function takes parameters of a callback function and a command, wraps them in the `RWCollectable`-derived classes, and adds the pair to the dictionary

```
void XCmdDispatcher::InsertCmd(
    CMDCallback theCallback,
    long theCommand,
    BOOLEAN theNewReplacesExistingCallback)
{
    CLongRWC* aKey = new CLongRWC( theCommand );
    if (theNewReplacesExistingCallback)
        itsCallbackTable.removeAndDestroy( aKey );

    itsCallbackTable.insertKeyAndValue( aKey,
        new XCMDCallbackRWC( theCallback ));
}
```

Advanced XVT-Power++

Slide 9

Dispatcher Implementation (cont.)

- ❖ `DispatchCmd()` takes a command, the data passed with the command, and an object on which to invoke the callback.

- ♦ The return value indicates if a callback function was found

```
BOOLEAN XCmdDispatcher::DispatchCmd(CView *theObject,
    long theCommand, void *theData) const
{
    // Find callback for this command
    CLongRWC aKey(theCommand);
    XCMDCallbackRWC *aCallback =
        (XCMDCallbackRWC*)itsCallbackTable.findValue(&aKey);
    if (! aCallback) return FALSE;

    // if a callback was located, invoke the method
    (theObject->*aCallback->value())(theCommand, theData);
    return TRUE;
}
```

Advanced XVT-Power++

Slide 10

Setting Up a Dispatcher

❖ To use the command dispatcher

- ♦ Declare and implement the callback functions
- ♦ Add a command dispatcher data member to your window class

```
class XMyWindow : public CWindow
{
public:
    ... standard methods ...
    // Callbacks:
    void TypeCheckInt(long, void*);
    void TypeCheckDate(long, void *);
    void AuthorizeUser(long, void *);
    void DenyRequest(long, void *);
    void ShowFinancials(long, void *);
protected:
    XCmdDispatcher *itsDispatcher;
};
```

Advanced XVT-Power++

Slide 11

Initializing the Dispatcher

❖ Instantiate the dispatcher and add callbacks to it in the window class constructor or initializer

```
XMyWindow::XMyWindow(...)
{
    ...
    // Create dispatcher and install callbacks:
    itsDispatcher = new XCmdDispatcher;

    itsDispatcher->InsertCmd(TypeCheckInt,    EDITIntCmd);
    itsDispatcher->InsertCmd(TypeCheckDate,  EDITDateCmd);
    itsDispatcher->InsertCmd(AuthorizeUser,  AUTHORIZECmd);
    itsDispatcher->InsertCmd(DenyRequest,    GETFinancialsCmd);
    ...
}
```

Advanced XVT-Power++

Slide 12

Using the Dispatcher

- ❖ The window's DoCommand changes from a switch statement to a table lookup. Any command not registered receives the default behavior

```
void XMyWindow::DoCommand(long theCommand, void *theData)
{
    if (itsDispatcher->DispatchCmd(this, theCommand, theData))
        return;

    CWindow::DoCommand( theCommand, theData );
}
```

Advanced XVT-Power++

Slide 13

Run-Time Changes

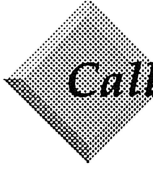
- ❖ The lookup table allows the application to dynamically change the way in which commands are handled

- ♦ When the user's access privileges change, for example

```
XMyWindow::AuthorizeUser( long theCommand, void* theData )
{
    if ( ... the secret password was entered ... )
        // Start honoring requests to see private information
        itsDispatcher->InsertCmd(ShowFinancials, GETFinancialsCmd);
}
```

Advanced XVT-Power++

Slide 14



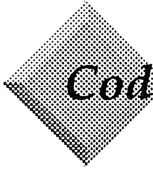
Callback Table Summary

- ❖ Uses a dictionary to associate a callback function with a command
- ❖ Derive wrapper classes from RWCollectable to store the command and callback function in the dictionary
- ❖ Use this technique to handle large numbers of commands, or to allow for changing the way a command is handled at runtime
- ❖ Remember, this is not limited to a CWindow implementation or callbacks on a CView object



Advanced XVT-Power++

Slide 15



Code Walkthrough

- ❖ Implement a lookup table for commands



Advanced XVT-Power++

Slide 16

Commands Needing a Reply

- ❖ In many instances, an object sends a command because it needs information from another object
- ❖ To decouple the two objects, write a simple command class
 - ♦ Identify the object expecting a reply
 - ♦ Identify the command that should be used to send the reply
 - ♦ Deriving from RWCollectable is useful if you need to maintain history lists (to implement Undo, for example)

Advanced XVT-Power++

Slide 17

A Class for Requesting a Reply

```
class XCommand : public RWCollectable
{
    RWDECLARE_COLLECTABLE( XCommand );
public:
    XCommand( long theCallbackCommand = NULLcmd,
              CBoss* theSender        = 0,
              void* theData           = 0 );
    virtual ~XCommand() {}

    void SendReply( void* theReplyData ) const
    { itsSender->DoCommand( itsCmd, theReplyData ); }
    long GetCommand( void ) const { return itsCmd; }
    void* GetData( void ) const   { return itsData; }
protected:
    long    itsCmd;
    CBoss*  itsSender;
    void*   itsData;
};
#define XCommandId 31201
```

Advanced XVT-Power++

Slide 18

Requesting a Reply

```
void XEmployeeFormWin::DoCommand(long theCommand,void* theData)
{
    switch (theCommand)
    {
        case NULLcmd:           // stop "undefined" commands
        case NATIVEViewCmd:      // only of interest to CScroller
            break;

        case SEARCHcmd:
        {
            XEmployee *aPattern = ReadEmployeePattern( TRUE );
            if (aPattern) {
                XCommand aCommand( RESULTScmd, this, aPattern );
                CWindow::DoCommand( theCommand, &aCommand );
            }
        } break;
        ....
    }
}
```

Advanced XVT-Power++

Slide 19

Responding to the Reply

```
void XEmployeeFormWin::DoCommand(long theCommand,void* theData)
{
    switch (theCommand)
    {
        ...
        case RESULTScmd:
        {
            XSearchReply* aReply = (XSearchReply*)theData;
            if (aReply->itsMatchCount != 0) {
                IEmployeeFormWin( aReply->itsEmployee,
                                aReply->itsMatchCount,
                                aReply->itsIndex );
                itsPreviousReply = *aReply;
            }
            else
                xvt_dm_post_note( "No matching employees found" );
        } break;
        ....
    }
}
```

Advanced XVT-Power++

Slide 20

Broadcasting

- ❖ **Broadcasting is a technique similar in spirit to the built-in Automatic Data Propagation mechanism**
 - ◆ Broadcasting is more limited, but requires less programming to use
 - ◆ Broadcasting must follow the object hierarchy, whereas ADP can be used to notify many unrelated objects
- ❖ **Broadcasting uses downward chaining to notify all sub-objects in the hierarchy**
 - ◆ This is particularly useful when multiple windows are displaying the same data

Advanced XVT-Power++

Slide 21

Implementing Broadcasting

```
void XEmployeesDoc::DoCommand(long theCommand, void* theData)
{
    switch (theCommand)
    {
        case NEXTEmployeeCmd:
        {
            XSearchReply* aMsg = (XSearchReply*)theData;
            if (NextEmployee( aMsg ) {
                RWindowList* aWindowList = GetSubObjects();
                RWindowIterator nextWindow( *aWindowList );

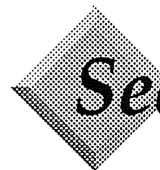
                CBoss* anObject;
                while (nextWindow()) {
                    anObject = (CBoss*)nextWindow.key();
                    anObject->DoCommand( RESULTScmd, aMsg );
                }
            } break;
            ....
        }
    }
}
```

Advanced XVT-Power++

Slide 22

Section 7

Printing Essentials



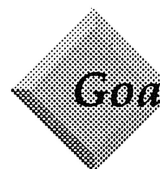
Section 7

Printing Essentials



Advanced XVT-Power++

Slide 1



Goals of Section 7

- ❖ Understand the printing process
- ❖ Review default printing behaviors of the Power++ application framework
- ❖ Learn key techniques for successful printing



Advanced XVT-Power++

Slide 2

The Printing Process

- ❖ When the user asks your application to print, three important steps occur
 - ♦ Your application first fills an internal print queue with the pages to be printed.
 - ♦ A special print window is created.
 - ♦ Each page “draws” itself on the print window. The result is a printed page on the printer.
- ❖ The owner of the print queue is a global object called the Print Manager

Advanced XVT-Power++

Slide 3

Print Manager

- ❖ CPrintManager owns and manages the queue of pages of a print job
- ❖ Each “page” of a print job is some CView object
 - ♦ Often each window is represented by a page in the queue
- ❖ After all the pages are loaded into the queue, invoke the Print Manager’s DoPrint() method
 - ♦ Creates the print window
 - ♦ Instructs each page to draw itself on the print window

```
CPrintMgr* printMgr = G->GetPrintMgr();  
CWindow* aWin;  
RWOIterator nextWindow(G->GetDesktop()->GetWindows());  
while (aWin = (CWindow*) nextWindow())  
    printMgr->Insert(aWin, aWin->GetFrame());  
BOOLEAN ok = printMgr->DoPrint(GetPrintRecord());
```

Advanced XVT-Power++

Slide 4

Printing Interface

- ❖ Every view has a standard method that the Print Manager calls to notify the view to print itself

```
virtual void DoPrintDraw(const CRect& theRegion);  
virtual void PrintDraw(const CRect& theRegion);
```

- ❖ DoPrintDraw() is the downward chaining, wide interface method

- ♦ Tells a CUnits object, *if any*, to map to the page dimensions rather than the screen dimensions
- ♦ Calls PrintDraw() on itself
- ♦ Chains the DoPrintDraw() to all nested views
- ♦ When finished, it resets the units to draw to the screen

- ❖ The default implementation of PrintDraw() is to call Draw()

Advanced XVT-Power++

Slide 5

Built-In Behavior

- ❖ Which part of the CView hierarchy provides an implementation of the Draw() function that will also print the view?

- ♦ Every CSubview class
- ♦ The CText class
- ♦ Wire frames

- ❖ Which views are not printable by default ?

- ♦ All native views
- ♦ The CCTable class

- ❖ The printing behavior of CNativeTextEdit and CVirtualFrame may not be sufficient

- ♦ Only the contents that appear on the screen will be printed

Advanced XVT-Power++

Slide 6

Mapping to Printer Dimensions

- ❖ The default implementation converts to printer coordinate mapping before telling an object to draw itself on the printer

```
if (itsUnits)
    itsUnits->SetOutputDevice(PRINTER);

if (Prepare(theClippingRegion))
    PrintDraw(theClippingRegion);

if(itsUnits)
    itsUnits->SetOutputDevice(SCREEN);
```

- ❖ BUT - the default framework behavior uses a 1-1 mapping to convert logical units to device units
 - ♦ No CUnits object is created by default!

Advanced XVT-Power++

Slide 7

The Key to Successful Printing

- ❖ To print successfully:
 - ♦ Create a CUnits object
 - ♦ Instruct the CUnits to dynamically change the mapping based on the output device

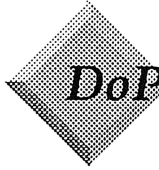
```
void XMyApp::StartUp(void)
{
    // NOTE: The base class method must be called before
    // adding any of your own code.
    CApplication::StartUp();

    CUnits* aUnits = new CUnits;
    aUnits->SetDynamicMapping( TRUE );
    SetUnits( aUnits );

    // Create default documents and their initial windows
    DoNew();
}
```

Advanced XVT-Power++

Slide 8



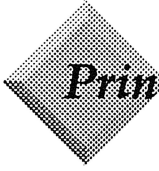
DoPrint() Functionality

- ❖ The application framework has default behavior implemented when the standard File | Print menu item is selected
 - ♦ A CDocument sends each open window owned by the document to the printer, each as its own page
 - ♦ A CApplication object sends each open window in the application to the printer, each as its own page
- ❖ Typically you will want to override this behavior at the window level or the document level
 - ♦ Also, remember to assign a menubar to the window - Why?



Advanced XVT-Power++

Slide 9



Printing Essentials: A Summary

- ❖ Create a CUnits object if your application needs to print
- ❖ Override the DoPrint method at the window or document level of the object hierarchy
 - ♦ Assign a menubar to your window so the notification goes through the expected path in the object hierarchy
- ❖ Native controls are not intended to be printable
- ❖ Still to come...
 - ♦ Pagination
 - ♦ How to print native views
 - ♦ Report generation
 - ♦ Adornments - page headers and footers

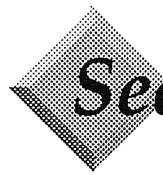


Advanced XVT-Power++

Slide 10

Section 8

Data and Document Management



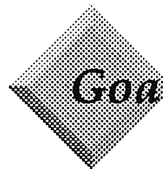
Section 8

Data and Document Management



Advanced XVT-Power++

Slide 1



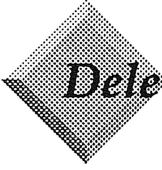
Goals of Section 8

- ❖ **Change the default behavior for deleting documents**
- ❖ **Define document classes for database applications**
- ❖ **Examine techniques for form-based applications**



Advanced XVT-Power++

Slide 2



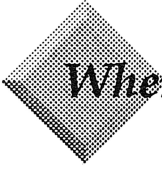
Deleting Document Objects

- ❖ The default behavior in Power++ is to delete all the document objects when the application object is deleted - i.e. at application termination
- ❖ If you want documents to be deleted sooner, you must program this yourself
 - ♦ When do you delete a document?
 - ♦ Which object is responsible for doing the deletion?
 - ♦ How do you do accomplish this?



Advanced XVT-Power++

Slide 3



When a Window Closes...

- ❖ When a window is closed, it gets deleted and its destructor gets called
- ❖ Basic framework concepts
 - ♦ Every document manages a list of windows it owns
 - ♦ Every window knows it belongs to a document
- ❖ One thing the window destructor does is to remove this window from the list managed by its document

```
CWindow::~CWindow(void)
{
    ...
    itsDocument->RemoveWindow(this);
}
```



Advanced XVT-Power++

Slide 4

Detecting Last Window Closed

- ❖ **Override RemoveWindow() in your document class. Check after each call to determine if the list of windows is empty**

```
XMyDoc::RemoveWindow(CWindow* theWindow)
{
    CDocument::RemoveWindow(theWindow);
    if (itsWindows->entries() == 0) {
        ... do something ...
    }
}
```

Advanced XVT-Power++

Slide 5

Signaling A Dead Document

- ❖ **The document cannot delete itself. Instead, it should notify the application to delete this document**

```
XMyDoc::RemoveWindow(CWindow* theWindow)
{
    CDocument::RemoveWindow(theWindow);
    if (itsWindows->entries() == 0)
        DoCommand( DELETEThisDocCmd, this );
}
```

Advanced XVT-Power++

Slide 6

Deleting the Document

- ❖ When the application receives the DoCommand, it can't delete the Document immediately. Instead, it adds the document to a to-do list

```
XMyApp::DoCommand(long theCommand, void* theData)
{
    switch( theCommand ) {
        case DELETEThisDocCmd:
            itsDeadDocs.append( (CDocument*)theData );
            ... is this sufficient? ...
            break;
    }
}
```

Advanced XVT-Power++

Slide 7

Deleting the Document (cont.)

- ❖ The application needs to send itself a reminder to check its to-do list

- ♦ Use an interval timer from the PTK

```
XMyApp::DoCommand(long theCommand, void* theData)
{
    switch( theCommand ) {
        case DELETEThisDocCmd:
            itsDeadDocs.append( (CDocument*)theData );
            xvt_timer_create( TASK_WIN, 100 );
            break;
    }
}
```

Advanced XVT-Power++

Slide 8

Finally, Deleting the Document

- ❖ When the interval expires the application is notified through its DoTimer method. It cancels the timer and deletes all documents in its to-do list

```
XMyApp::DoTimer(long theTimerId)
{
    xvt_timer_destroy( theTimerId );
    itsDeadDocs.clearAndDestroy();
}
```

Advanced XVT-Power++

Slide 9

A Race Condition

- ❖ At application shutdown, there is a possibility of a race condition: both the application destructor and the interval timer logic will attempt to delete the documents.
 - ◆ How do you safely clear out two lists that may contain references to the same document objects?

Advanced XVT-Power++

Slide 10

Application Destructor

❖ Use RWCollection features to solve the race condition problem

- ❖ RogueWave guarantees that if an object appears multiple times in a collection, the clearAndDestroy method will only delete it once

```
XMyApp::~XMyApp(void)
{
    itsDeadDocs += itsDocuments; // combine the two lists
    itsDeadDocs.clearAndDestroy();
}
```

Advanced XVT-Power++

Slide 11

Document Design for Databases

❖ Consider an employee database application. The typical user interactions

- ❖ The user logs in to the database to start a session
- ❖ Several types of information are stored about each employee:
 - Personal data, job and salary data, notes, etc
- ❖ The user may want to
 - add new employees, or update existing employees
 - find a set of employees that match specific criteria
 - print reports on a specific employee or a set of employees
- ❖ Different users may have different access rights to data
- ❖ At the end of a session the user logs out

❖ How should this application be structured?

Advanced XVT-Power++

Slide 12

Database Form

Employee ID:		1065
Last Name:	Doe	Search
First Name:	John	Clear
Middle Name:		
Position:	Consulting Engineer	>>
Date of Hire:	06/18/93	<<
Street Address:	6 West Fifth Ave	
City:	Longmont	
State:	CO	Postal Code: 80207
Phone:	303 - 445 - 8125	
OK		Cancel

Record 1 of 15

Advanced XVT-Power++

Slide 13

An Employee Database

- ❖ One overriding data concept is that of employee information - the derived document class should encapsulate a collection of employees
 - ♦ Each different type of information on the employee may be represented by a different window class, but each window class will show a subset of the total employee information
 - ♦ Avoid writing separate document classes for different components of the employee information
- ❖ Another concept needed is that of a result set
 - ♦ A result set contains the subset of employees matching the last specified criteria
 - ♦ This is a subset of the same data - should each result set be its own document, or should it be part of a single document class?

Advanced XVT-Power++

Slide 14

Databases (cont.)

- ❖ **The other concept needed is that of a login and access rights mechanism**
 - ♦ This could be a part of the same data class, or it could be a separate, light-weight data class - which will work better?
- ❖ **Design considerations**
 - ♦ A separate login/access-rights document class encapsulates the concepts more clearly. It's primary job is to manage users of the application, not employees.
- ❖ **Handling the login and database connection**
 - ♦ The login screen should probably be part of application StartUp behavior.
 - ♦ Application ShutDown should make sure the database connection is closed

Advanced XVT-Power++

Slide 15

Implementation Hints

- ❖ **The employee database can use many techniques described in previous sections**
 - ♦ Derive an RWCollectable to encapsulate employee information
 - ♦ Manage access rights by initializing a command callback table
 - ♦ Search, Next Record and Previous Record operations can be implemented using commands that require a reply
 - ♦ If multiple windows display information on the same employee simultaneously, consider broadcasting the reply
 - ♦ RWRegexp could be used to locate a set of employees matching specific criteria
 - ♦ Write an initializer function for any form-style windows, to update all the fields on each new record
- ❖ **Printing is discussed in the next section**

Advanced XVT-Power++

Slide 16

Code Walkthrough

- ❖ Review the code for the employee database application

 **Advanced XVT-Power++**

Slide 17

Section 9

Specialized Printing

1. *Introduction*

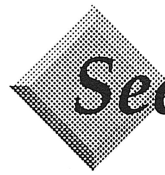
2. *Method*

3. *Results*

4. *Discussion*

5. *Conclusion*

6. *References*



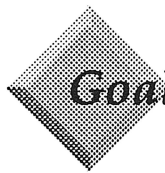
Section 9

Specialized Printing



Advanced XVT-Power++

Slide 1



Goals of Section 9

- ❖ Understand how to implement report generation and other forms of pagination
- ❖ Learn how to print native views
- ❖ Develop techniques for page formatting - headers, footers and margins



Advanced XVT-Power++

Slide 2

Pagination

- ❖ **Some views have a need for pagination**
 - ◆ `NScrollText`, `CCTable`, `CVirtualFrame`,...
- ❖ **Report generation requires converting a form-based interface displaying many records to one or more pages of tabular data**
- ❖ **CPrintManager requires that each page placed in the queue be represented by a view**
- ❖ **How can we satisfy all these constraints?**

Advanced XVT-Power++

Slide 3

Paginating a Scrolling View

- ❖ **To paginate a scrolling view**
 - ◆ Write a `DoPrint` function for the view class that splits the entire virtual frame into page size chunks. Insert the same view into the `PrintManager` multiple times, clipped to different regions
 - ◆ Write a `DoPrintDraw` function for the view that computes a new coordinate translation each time a new page starts. The translation is used by the `Draw` function to ensure each page starts at 0, 0 on a sheet of paper, even though the views are at different offsets from the virtual frame
 - ◆ Modify your `Draw` function to account for the translation

Advanced XVT-Power++

Slide 4

Paginate a Scrolling View (cont.)

❖ DoPrint first calculates a page dimension

```
BOOLEAN XHierarchyBrowser::DoPrint(
    const CRect& /*theRegion*/ ) const
{
    long aPrintHeight, page_width, vres, hres;
    #if (XVTWS == MTFWS)
        aPrintHeight = xvt_vobj_get_attr( NULL_WIN,
                                           ATTR_PRINTER_HEIGHT );
        ... get other information if needed ...
    #else
        xvt_app_escape( XVT_ESC_GET_PRINTER_INFO,
            G->GetApplication()->GetPrintRecord(),
            &aPrintHeight, &aPageWidth, &hres, &vres );
    #endif
    int lines_per_page = (VLogical(aPrintHeight) /* -margins */
        / aLineHeight);
    UNITS aPageHeight = lines_per_page * aLineHeight;
    ....
}
```

Advanced XVT-Power++

Slide 5

Paginate a Scrolling View (cont.)

❖ DoPrint splits the contents into multiple pages

```
BOOLEAN XHierarchyBrowser::DoPrint(
    const CRect& /*theRegion*/ ) const
{
    ... calculate page height ...

    // break the virtual frame into regions of page dimensions
    CPrintMgr* printMgr = G->GetPrintMgr();
    CRect aPrintFrame = GetVirtualFrame();
    UNITS aCutoff = aPrintFrame.Bottom();
    aPrintFrame.Bottom( aPageHeight );

    do {
        printMgr->Insert( this, aPrintFrame );
        aPrintFrame.Top( aPrintFrame.Bottom() );
        aPrintFrame.Bottom( aPrintFrame.Bottom() + aPageHeight );
    } while (aPrintFrame.Top() <= aCutoff);
    ....
}
```

Advanced XVT-Power++

Slide 6

- Journal of Management Inquiry* 18(6)

19. 100

W91-2, 1

10



72. 100%

Report

- ◆ Need to iterate over the records in a result set, creating an entry (typically a line) on the page for each record
- ◆ Many, many views would be created if each field in each entry is a view



Figure 1

Surrogates for Report Generation

- ❖ To implement report generation, surrogates (also known as proxies) are needed
 - ♦ Proxies represent another object, and provide access to it on demand
- ❖ For report generation, a page would be a subview containing proxies for each line on the page
 - ♦ The line proxy may be made up of field proxies
 - ♦ The page can also have header and footer views

Advanced XVT-Power++

Slide 9

Form Printing

- ❖ What if the application needs to print an image of a form?
 - ♦ Native views don't know how to print themselves
- ❖ How can we add the ability to print native views without modifying the framework?

Advanced XVT-Power++

Slide 10

A Form Printing Class

- ❖ To add the ability to print native views
 - ♦ Derive a CWindow class that implements a table of callback functions for printing views
 - ♦ Override DoPrintDraw to lookup and call the callback function, rather than calling DoPrintDraw on each nested view
 - ♦ Allow for instance callbacks as well as class callbacks
- ❖ This technique is not limited to native views

Advanced XVT-Power++

Slide 11

Form Printing Window Class

```
typedef void (*PrintFunction)( CView* theView,
                              const CRect& thePrintRegion );

class XFormWin : public CWindow
{
public:
    virtual ~XFormWin( void );
    virtual void DoPrintDraw( const CRect& theClippingRegion );
    BOOLEAN AddInstanceCallback( const CView* theView,
                                PrintFunction theCallback );
    BOOLEAN AddClassCallback( long theTypeId,
                              PrintFunction theCallback );
protected:
    // Abstract base class - protected constructor
    XFormWin(...);
private:
    RWHashDictionary itsCallbacks;
};
```

Advanced XVT-Power++

Slide 12

Form Printing Window Class

```
void XFormWin::DoPrintDraw( const CRect& theClippingRegion )
{
    PrintDraw( theClippingRegion );
    RWOIterator nextView( *itsSubviews );
    CView* aView;
    while (nextView()) {
        aView = (CView*)nextView.key();
        CLongRWC anInstanceKey(aView->GetId());
        CLongRWC aClassKey(aView->GetRunTimeTypeInfo()->GetId());
        XCollectableCallback* aCollectable;
        // Check lookup table for instance-specific PrintDraw function
        // If no instance-specific function, check for class-specific
        if ((aCollectable = itsCallbacks.findValue(&anInstanceKey)) ||
            (aCollectable = itsCallbacks.findValue(&aClassKey)))
            aCollectable->Callback()( aView, aDamageRegion );
        else
            // If neither, call the default function
            aView->DoPrintDraw( aDamageRegion );
    }
}
```

Advanced XVT-Power++

Slide 13

Registering Default Methods

```
XFormWin::XFormWin(CDocument *theDocument,
                  const CRect& theRegion,
                  const CStringRW& theTitle,
                  long theWindowAttributes,
                  WIN_TYPE theWindowType,
                  int theMenuBarId)
: CWindow( theDocument, theRegion, theTitle,
          theWindowAttributes, theWindowType, theMenuBarId )
{
    // register default class-specific printing functions
    AddClassCallback( NEditControlID, PrintNEditControl );
    AddClassCallback( NTextID, PrintNText );
}

static void PrintNText( CView* theCtl, const CRect& theRegion )
{
    CDrawingContext aDC( theCtl );
    CPoint aBaseLine( theCtl->GetGlobalFrame().Left() + 2,
                     theCtl->GetGlobalFrame().Bottom() );
    aDC.DrawText( theCtl->GetTitle(), aBaseLine );
}
```

Advanced XVT-Power++

Slide 14

Registering Instance Methods

```
XEmployeeFormWin::XEmployeeFormWin(CDocument *theDocument, ....)
: XFormWin(theDocument, theRegion, theTitle,
    theAttributes, theWindowType, theMenuBar)
{
    FORMPRNTFactory.DoCreateViews( this, FormWin, FALSE, &itsData );
    AddInstanceCallback( itsData.itsLastName, PrintLastName );
    GetMenuBar()->SetEnabled( M_FILE_PRINT, TRUE );
    GetMenuBar()->SetEnabled( M_FILE_PG_SETUP, TRUE );
}

static void PrintLastName(CView* theCtl, const CRect& /*theRegion*/)
{
    CDrawingContext aDC( theCtl );
    CPoint aBaseLineLeft( 0, theCtl->GetFrame().Height()-4 );
    aBaseLineLeft.Globalize( theCtl );
    aDC.DrawRect( theCtl->GetGlobalFrame() );
    aDC.DrawText( theCtl->GetTitle(), aBaseLineLeft );
}
```

Advanced XVT-Power++

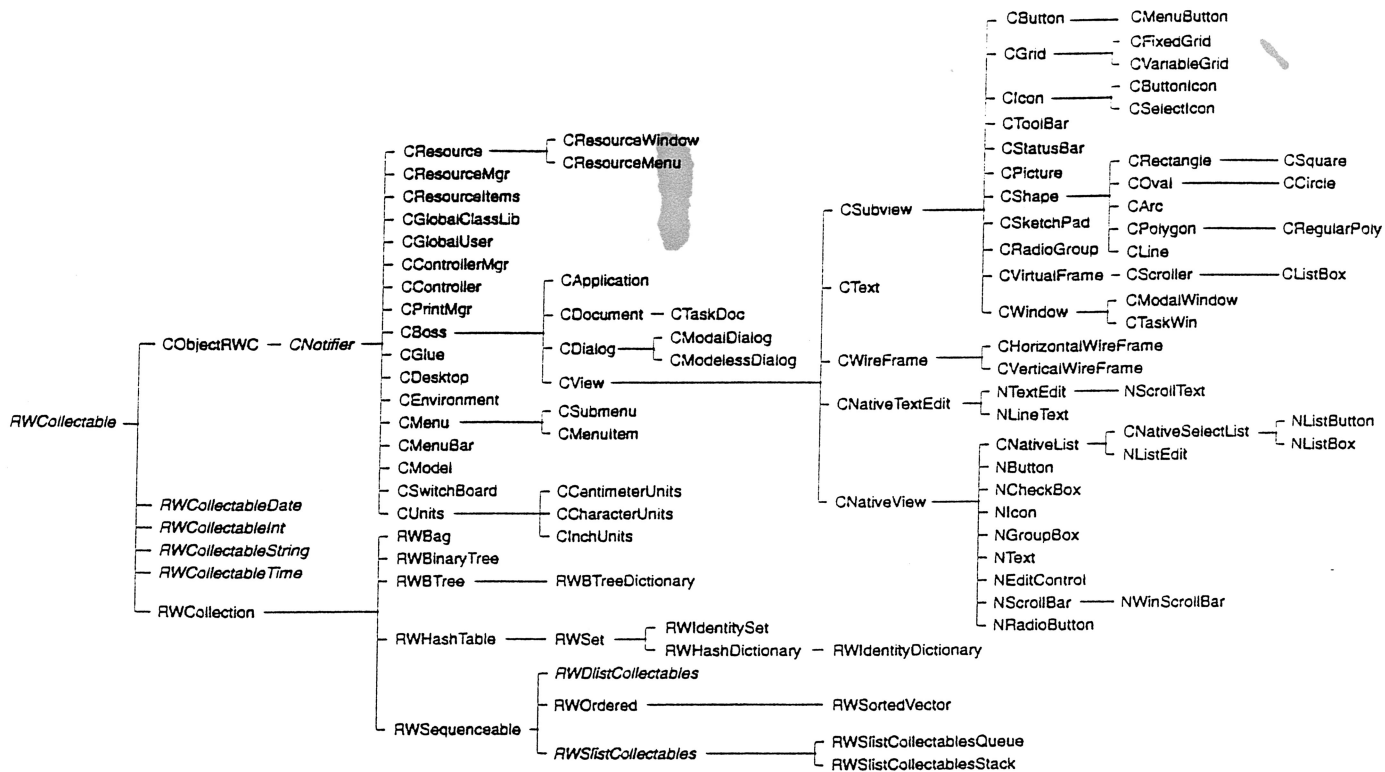
Slide 15

Any Questions?

Advanced XVT-Power++

Slide 16

XVT-Power++ 3.2 Hierarchy



Atomic

CBrush
CFloat
CFont
CImage
CPen
CPoint
CRect
RWBitVec
RWCString
RWCSUBSTRING
RWDate
RWInteger
RWTime
RWWString
RWWSubString

Generic

- `RWG8itVec(size)`
- `RWGDistlist(type)`
- `RWGDistlistiterator(type)`
- `RWGOrderedVector(val)`
- `RWGQueue(type)`
- `RWGSlist(type)`
- `RWGSlistiterator(type)`
- `RWGSortedVector(val)`
- `RWGStack(type)`
- `RWGVector(val)`

File

RWFile - RWFileManager
RWBTreeOnDisk
RWCachedManager

Template

- RWIsvDlink<T>
- RWIsvSlink<T>
- RWTBivVec<size>
- RWTIsvDlist<T>
- RWTIsvSlist<T>
- RWTPtrDlist<T>
- RWTPtrDlistIterator<T>
- RWTPtrHashTable<T>
- RWTPtrHashSet<T>
- RWTPtrHashTableIterator<T>
- RWTPtrHashDictionary<K,V>
- RWTPtrHashDictionaryIterator<K,V>
- RWTPtrOrderedVector<T>
- RWTPtrSlist<T>
- RWTPtrSlistIterator<T>
- RWTPtrSortedVector<T>
- RWTPtrVector<T>
- RWTStack<T,C>
- RWTQueue<T,C>
- RWTValDlist<T>
- RWTValDlistIterator<T>
- RWTValHashTable<T>
- RWTValHashSet<T>
- RWTValHashTableIterator<T>
- RWTValHashDictionary<K,V>
- RWTValHashDictionaryIterator<K,V>
- RWTValOrderedVector<T>
- RWTValSortedVector<T>
- RWTValSlist<T>
- RWTValSlistIterator<T>
- RWTValVector<T>
- RWTValVirtualArray<T>

iterators

```

CSParseArrayIterator
CSParseCollector
CSParseRowIterator

RWIterator
├── RWBagIterator
├── RWBinaryTreeIterator
├── RWDistCollectablesIterator
├── RWSetIterator - RWHashDictionaryIterator
├── RWOrderedIterator
└── RWSiistCollectablesIterator

RWIterator ─── CRevOrdIteratorRW

RWDListIterator ─── RWDistCollectablesIterator
RWSListIterator ─── RWSiistCollectablesIterator

```

Utilities

CStackable	CDrawingContext
CTypeInfo	
Error	
Global	Stream
Mem	streambuf
RWCRegexp	
RWCTokenizer	
RWBench	
RWErrObject	RWvics
RWFFactory	
RWModel	
RWModelClient	
RWTimer	
RWVirtualPageHeap	Lo
RWBufferedPageHeap	RW
RWDiskPageHeap	RW
RWTokenizer	RW

Stream

```

streambuf {
  RWCLIPstreambuf
  RWDDestreambuf
}

RWwios {
  RWwistream {
    RWbistream
    RWpistream
    RWXDRistream
  }
  RWwostream {
    RWbostream
    RWpostream
    RWXDRostream
  }
}

```

Locate

```

RWLocale { RWLocaleDefault
           RWLocaleSnapshot
RWZone  — RWZoneSimple

```

Collectables

```

CFloat      ----- CFloatRWC
CNotifier   -----
CPoint      ----- CPointRWC
CNotifier   -----
CStringCollection ----- CStringCollectionRWC
CNotifier   -----
CRect       ----- CRectRWC
CNotifier   -----
RWCString   ----- CStringRW
CStringRW   ----- CStringRWC
CNotifier   -----
RWDate      ----- RWCollectableDate
RWCollectable -----
RWInteger   ----- RWCollectableInt
RWCollectable -----
RWCString   ----- RWCollectableString
RWCollectable -----
RWTime      ----- RWCollectableTime
RWCollectable -----

```

Collections

```

CSparseArray
CStringCollection
RWDList ----- RWDlistCollectables
RWSList ----- RWSlistCollectables
RWSList ----- RWSlistCollectables

```

Legend: — indicates a derivative relationship, which proceeds from left to right
italics indicate multiple references

© 1995 XVT Software Inc., 4900 Pearl East Circle, Boulder, CO 80301, (303) 443-4223, FAX: (303) 443-0969

50637.1 • 4/98



XVT
SOFTWARE